NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF A SOFTWARE COMMUNICATION ARCHITECTURE FOR THE JANUS-3D VISUALIZER

by

Christopher S. Upson

September 1994

Thesis Advisor: David R. Pratt

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE DESIGN AND IMPLEMENTATION OF A SOFTWARE COMMUNICATION ARCHITECTURE FOR THE JANUS-3D VISUALIZER (UNCLASSIFIED)	5. FUNDING NUMBERS
6. AUTHOR(S) Upson, Christopher S.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT *(Maximum 200 words)*

During the National Guard's mobilization for Desert Shield/Desert Storm, deficiencies were noted in the command and control skills of battalion and brigade level units. The major problem addressed by this research was to improve these skills by developing a software communication architecture that would allow events occurring in two dimensions in the Janus Combat Modeler to be seen in three dimensions on a visualization tool called the Janus-3D Visualizer over both local ethernet and wide area telephonic networks. The challenge was to minimize network latency along with providing accurate data in order to maintain the time and space coherence of the simulation.

The approach taken was to first determine where the needed information resided in the Janus modeler. Next, a protocol was developed to extract the information and send it to the Janus-3D Visualizer over the local network for viewing. Finally, a protocol was developed to transmit this information over a telephonic network upon request in order for it to be viewed on a remote Janus-3D Visualizer.

As a result of this work, Janus battles can be viewed in three dimensions and in real time by brigade and battalion commanders and their staffs without them having to spend valuable training funds to move everyone to a single location. National Guard units can now use the Janus modeler more often and more realistically in order to improve command, control and communication skills.

14. SUBJECT TERMS Wide Area Network, Local Area Network, Communication Architecture, Software, Ethernet, Telephone, Janus Combat Modeler	15. NUMBER OF PAGES 77
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

DESIGN AND IMPLEMENTATION OF A SOFTWARE COMMUNICATION ARCHITECTURE FOR THE JANUS-3D VISUALIZER

Christopher S. Upson
Captain, United States Army
B.B.A., Siena College, 1985

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

Accesion For	
NTIS CRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

Author: [illegible]
Christopher S. Upson

Approved By: [illegible]
David R. Pratt, Thesis Advisor

[illegible]
Gilbert M. Lundy, Second Reader

[illegible]
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

During the National Guard's mobilization for Desert Shield/Desert Storm, deficiencies were noted in the command and control skills of battalion and brigade level units. The major problem addressed by this research was to improve these skills by developing a software communication architecture that would allow events occurring in two dimensions in the Janus Combat Modeler to be seen in three dimensions on a visualization tool called the Janus-3D Visualizer over both local ethernet and wide area telephonic networks. The challenge was to minimize network latency along with providing accurate data in order to maintain the time and space coherence of the simulation.

The approach taken was to first determine where the needed information resided in the Janus modeler. Next, a protocol was developed to extract the information and send it to the Janus-3D Visualizer over the local network for viewing. Finally, a protocol was developed to transmit this information over a telephonic network upon request in order for it to be viewed on a remote Janus-3D Visualizer.

As a result of this work, Janus battles can be viewed in three dimensions and in real time by brigade and battalion commanders and their staffs without them having to spend valuable training funds to move everyone to a single location. National Guard units can now use the Janus modeler more often and more realistically in order to improve command, control and communication skills.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. OBJECTIVE

Our objective for this thesis was to develop a software communications architecture for the Janus-3D Visualizer that is robust and efficient. The Janus 3D Visualizer is a program that displays in 3 dimensions what is occurring in a Janus Combat Modeler scenario in real time [VAGL94]. The success of this project was measured by the ability of the 3D Visualizer to accurately and efficiently depict a simultaneously running battle on Janus over both a local area network and wide area network without communication failure or significant communication delay.

B. BACKGROUND

Military budgets continue to shrink significantly. With these shrinking budgets come fewer opportunities for military personnel to train on their war time equipment as such training is costly in both time and logistical resources. This reality has, in recent years, led to the rapidly growing research, development and use of simulators to train our nation's military. These simulations are not meant to replace traditional hands-on training, but to supplement it in order to enhance its benefit as opportunities to conduct traditional training are far fewer than in the past.

Because of this need for low cost technological solutions, the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School is conducting research on 3D visual simulators that use commercial off-the-shelf graphics workstations. The primary research project of the laboratory is NPSNET, a three-dimensional, real-time, networked vehicle simulator [PRAT93]. A principle focus of this work is the development of software to create and run realistic visual simulations on low cost hardware, thus saving the user money while still providing a product that can be used for effective training. This is also the goal of the Janus-3D Visualizer project, of which this work is a part.

C. MOTIVATION

In fiscal year 1992, the Simulation in Training for Advanced Readiness (SIMITAR) program was initiated as the result of a Congressional mandate. The purpose of this program is to improve the training of National Guard Roundout Brigades through the use of advanced technology. This program came about as a result of training readiness deficiencies that were identified during the Desert Shield/Desert Storm mobilization. [FUNK94]

The manager of this program is the Advanced Research Projects Agency (ARPA). ARPA is an organization whose principle mission is research and development in areas of advanced technology in order to find ways in which the Department of Defense (DOD) can better accomplish its missions. One of its primary goals is to improve cost and performance within the DOD.

It was noted during the Desert Shield/Desert Storm mobilization that the National Guard was far more proficient in small unit skills than it was in battalion and brigade level skills. This is partially attributable to the fact that it is much easier for the companies and platoons to get together to train than it is for the higher level commands, which are often separated by large distances. Therefore, getting these larger commands together for training is very expensive and thus usually occurs only during the annual two week training period. As a result of these observations, one of the primary goals of SIMITAR is to improve brigade and battalion staff battlefield synchronization skills by designing technologies to distribute training opportunities as far as possible. Increasing the quantity and quality of higher command training in the National Guard through 3 dimensional visualization and distribution is the purpose behind the Janus-3D Visualizer project.

The 3D Visualizer will provide commanders with a three-dimensional perspective view of a Janus battle in real time. This 3D versus 2D view will allow commanders to better position their units before and during the battle and to better critique their junior leaders both during and after the battle. The wide area communications capabilities of this project will allow the Janus battle to be viewed over a greater area, thus enabling a brigade and its battalions to participate in the same battle from remote locations. This will save significant costs as the brigade will not have to travel to a single location to conduct this type of training. Each battalion and the brigade headquarters can remain at their drill locations for the duration of the exercise. Therefore, the training can be conducted much more frequently. It is hoped that allowing National Guard brigade and battalion commanders and their staffs to view their own and each others actions in the battle in 3D as frequently as they desire will improve their command, control and communication skills and thus their overall readiness.

D. ORGANIZATION OF THESIS

This thesis is organized into six chapters. This chapter provides the background and motivation of the work performed. Chapter II provides an overview of the Janus Combat Modeler itself along with brief descriptions of previous works whose ideas and philosophies contributed to this project. An overview of the Janus-3D Visualizer both in a broad sense and then, more specifically, its communications requirements and functionality is provided in Chapter III. Chapter IV discusses the development of the local area network communications architecture of the Janus-3D Visualizer and its interaction with the Janus Combat Modeler as a scenario takes place. The wide area network communications architecture for the Janus-3D Visualizer is described in Chapter V. Chapter VI provides information on the run-time performance of the 3D Visualizer with respect to communications along with our conclusions and topics for future research. Finally, the Appendix consists of the Janus-3D Visualizer's user's guide.

II. OVERVIEW OF JANUS AND PREVIOUS WORK

A. THE JANUS COMBAT MODELER

The Janus Combat Modeler is a brigade and lower level unit computer based war-gaming simulation used widely by the United States Army. It has two primary missions. One is combat development and analysis of new doctrine, strategy and technology. The other, which is most closely related to the Janus-3D Visualizer project, is to provide commanders and their staffs an inexpensive method of training to improve their command, control and communications skills "while developing and testing realistic operation plans." [JANU93b]

1. History

The Janus simulation was first fielded in 1978 by the Lawrence Livermore National Laboratories (LLNL) to model nuclear effects and became known as Janus(L). It quickly gained notoriety for its innovative graphical user interface design and was thus acquired by the U.S. Army Training and Doctrine Command (TRADOC) Analysis Command (TRAC) located at White Sands Missile Range (WSMR) as part of the Janus Acquisition and Development Project in 1980. TRAC-WSMR modified and further developed Janus(L) into a simulation tool to support Army combat development and referred to this new version as Janus(T). [WALT92].

Both versions of Janus gained in popularity and became widely distributed to a varied user community. With this growing popularity, the Army recognized that Janus could also provide a valuable training tool in addition to its combat development role. As a result, the Janus(Army) Program was started in 1989 to develop one standardized version of Janus that could be used by both the combat development and training communities. This standardized version of the combat modeler is now known as simply Janus and its continued development is the responsibility of TRAC-WSMR. [WALT92] [JANU93b]

The original version of Janus was developed on Digital VAX computers running the VMS operation system and used Tektronix Model 4225 graphics workstations for display. In 1992, Janus was ported to the UNIX operating system with X-Windows workstations for graphics display. UNIX version 3.17 was used for the Janus-3D Visualizer project. [JANU93b]

2. Description

The Janus combat modeler is a monolithic simulation that is "two-sided, closed, interactive and stochastic." Monolithic refers to the characteristic that the Janus scenario actually runs on only one machine while a number of other systems display the forces. The model is termed two-sided because it allows for the simulation of two opposing forces, blue and red, that are simultaneously controlled by two sets of players on separate monitors. Each monitor displays only those forces on its side along with opposing forces that its forces have detected. In other words, a player on one side does not know the complete distribution of the opposition's forces. Janus is considered closed because it is a stand alone simulation system. No outside systems influence it in any way. Janus is also interactive as the operators at each workstation control the actions of their forces in real time as the scenario unfolds. This detailed control is possible as the model focuses on individual fighting system engagements. The results of these engagements occur according to the laws of probability and chance that are built into the modeler. Therefore, Janus is referred to as stochastic and may lead to different results each time a specific scenario is executed. [JANU93] Finally, Janus also has a replay capability and analyst workstation that allow for after action reviews of the scenario in terms of troop movements, artillery fires and individual weapon system hits and kills. As such, the Janus combat modeler provides many valuable insights for analysis and training purposes. [JANU93b]

The terrain database for the Janus battles is developed from digitized terrain elevation data (DTED) provided by the Defense Mapping Agency (DMA). The terrain is displayed in a way that is familiar to military users with its contour lines, roads, rivers, vegetation and urban areas. These terrain features affect line of sight and vehicle movement just as real terrain does. Also, its symbology includes operational overlays, military map unit symbols and weapon system icons. [JANU93b]

3. Hardware

The open, or UNIX, version of Janus runs on a hardware suite that consists of a Hewlett-Packard (HP) Model 715/50 host computer that is linked to up to 24 workstations [JANU93c]. For the Janus-3D Visualizer project, the HP host was networked to a number of Sun Classic workstations for normal Janus operations and a Silicon Graphics Indigo 2 Extreme workstation for 3D visualization. Each workstation also included a 19-inch color monitor, keyboard, and three-button mouse.

4. Software

Janus includes sixteen executable programs written in FORTRAN. These programs can be divided based on their functionality into four major groups: data base management; scenario creation and execution; scenario results analysis; and utility programs. User menu programs written in the Digital Command Language (DCL) are also included. These DCL programs allow easy access to the executables. [JANU93]

a. Database Management

Five programs are used to create and maintain the weapon systems, graphics symbols and terrain databases. Weapon systems characteristics and weather conditions are managed by the CSDATA program prior to their application to a scenario. Once this information has been read into a scenario, however, the TED program is used to edit it for that specific scenario. The SYMBOLS program, on the other hand, is used to modify the graphics symbols that are displayed on the terrain. The terrain itself is managed by the TRNFLTR and TED programs. The TRNFLTR program allows for the selection of part of a master terrain and its conversion into a format that is compatible with Janus. Finally, the

TED program is used to create terrain features such as contour lines, roads and rivers. [JANU93].

b. Scenario Creation and Execution

Seven programs are used to develop and execute the Janus scenarios. First, the FORCE program is used to enter the actual units into new scenarios and edit forces in existing scenarios. The MERGE program allows the user to combine the opposing forces of different scenarios to form a new scenario. The battlefield environment of a scenario can be initialized or edited by the INITSCEN program. Scenario specific information and database errors are written to a file by VFYSCEN in order to allow for error correction by the user. GRFVFY, on the other hand, only prints such information to the screen. The CONWOR, or controller workstation, program allows a workstation to be set up as a passive observer station allowing people to view the battle, but not to interact with it. Finally, the JANUS program allows for initial planning by the user and then the actual execution of the scenario. [JANU93]

c. Scenario Results Analysis

Scenario analysis is handled by two programs. The POSTP program generates numerous reports on the results of the battle such as direct and indirect fires and casualties caused by various weapon systems. The JAAWS program displays unit statuses for the user at certain intervals throughout scenario execution vice after execution as POSTP does. [JANU93]

d. Utility Programs

The utility programs consist of HELPEDIT and FORMS. HELPEDIT is used in creating and editing user help files. FORMS, on the other hand, is used to develop the screen forms for displaying scenario data. [JANU93]

B. PREVIOUS WORK

The following works were researched due to their networking implementations and teleprocessing methodologies. Both local area ethernet and wide area telephonic network implementations are discussed. Many of the ideas in these works were used in developing the software communications architecture for the Janus-3D Visualizer.

1. Distributed Virtual Environment System for Cooperative Work

Nobutatsu Nakamura, Keiji Nemoto and Katsuya Shinohara have performed research on distributed virtual environments over a network. They developed three techniques to allow them to create an immersive virtual world that can be shared by multiple users simultaneously across a network. Their techniques include a client/server communications method, a management mechanism for the virtual objects that inhabit the world, and a multi-process model to help them achieve flexibility across different system configurations. The technique most applicable to our project was their multi-process methodology. [NAKA94]

In their model, the main process of any client was the client manager process. It is this parent process that creates the numerous child processes necessary for their distributed virtual environment to function smoothly. In their system, child processes included those responsible for the network, graphics, 3D mouse and data-glove, among others. All of these processes could be created and killed independently of any other. This allowed for the many tasks that must occur simultaneously to be accomplished while also saving CPU memory by killing processes that were no longer needed. This capability is crucial to creating and executing believable distributed virtual environments. See Figure 1 for a depiction of their multi-process model.

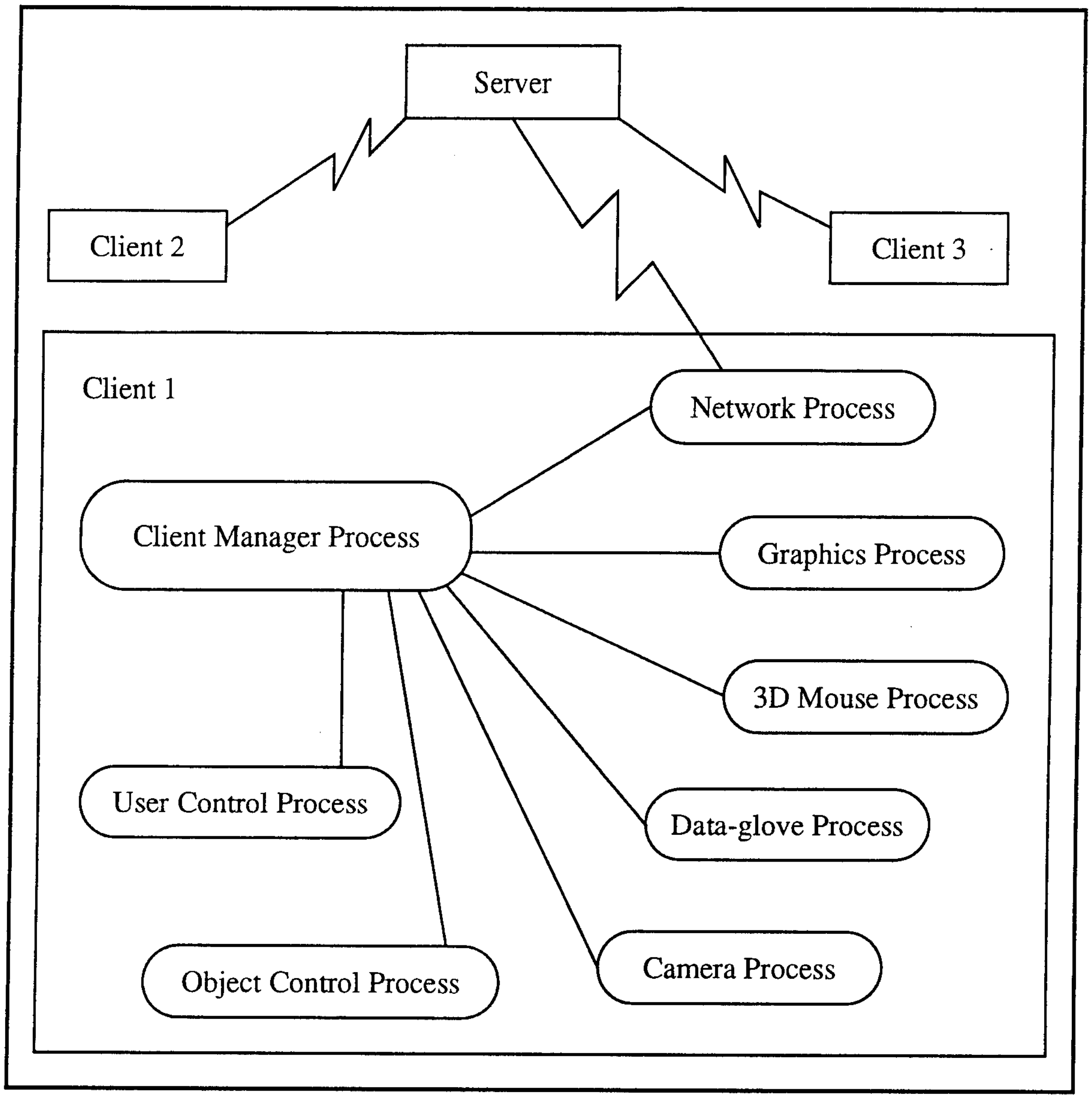


Figure 1. Multi-process Model for the Distributed Virtual Environment System

2. Lucasfilm's Habitat

Habitat was a networked virtual environment that allowed many players to participate in the world simultaneously. Each user played at his home from his own personal computer and connected to Habitat via modems and phone lines. This early attempt at designing a many user, networked virtual world entertainment system brought to light many issues that must be considered when doing so. The issues of concern to this project are those involving serial communications. In Habitat, they had to be able to allow the user a pleasing experience in the environment with the limitation of 300 BAUD connections. Therefore, bandwidth was a major concern. Only the absolutely essential data could be transmitted in order to achieve fast enough feedback to satisfy the user. In Habitat, they transmitted only behavioral data; changes in state. Presentation or modeling data could not be transferred in order to keep things moving fast enough to maintain user satisfaction. [MORN91] In this project, the same principles were followed in order to maintain the realism of the 3D display on the Janus-3D Visualizer.

3. Environmental Effects for Distributed Interactive Simulation (E2DIS)

E2DIS was an exercise conducted between the Naval Postgraduate School's (NPS) Computer Science Graphics and Video Laboratory and the Army's TEXCOM Experimentation Command (TEC) at Fort Hunter Liggett, California. Its purpose was to show the feasibility of the combination of a Virtual Reality (VR) interface with real world weapons testing. The real world events were displayed in the NPS Graphics Laboratory on NPSNET [ZYDA92]. NPSNET is a 3-dimensional (3D) battlefield simulator that operates simultaneously on separate workstations through the use of Distributed Interactive Simulation (DIS) network protocols [INST91]. The link between the real world events and NPSNET was accomplished with two modems transmitting formatted packets over a phone line from Fort Hunter Liggett to NPS. With the information received, NPSNET, using the Fort Hunter Liggett terrain, displayed a 3D animation of the actual test as it was occurring. See Figure 2 for a depiction of the flow of data across the E2DIS network. [PRAT94] This transmission of entity location information via phone lines over a wide area for subsequent display in 3D was very useful in the development of the wide area communications software architecture for the Janus-3D Visualizer.

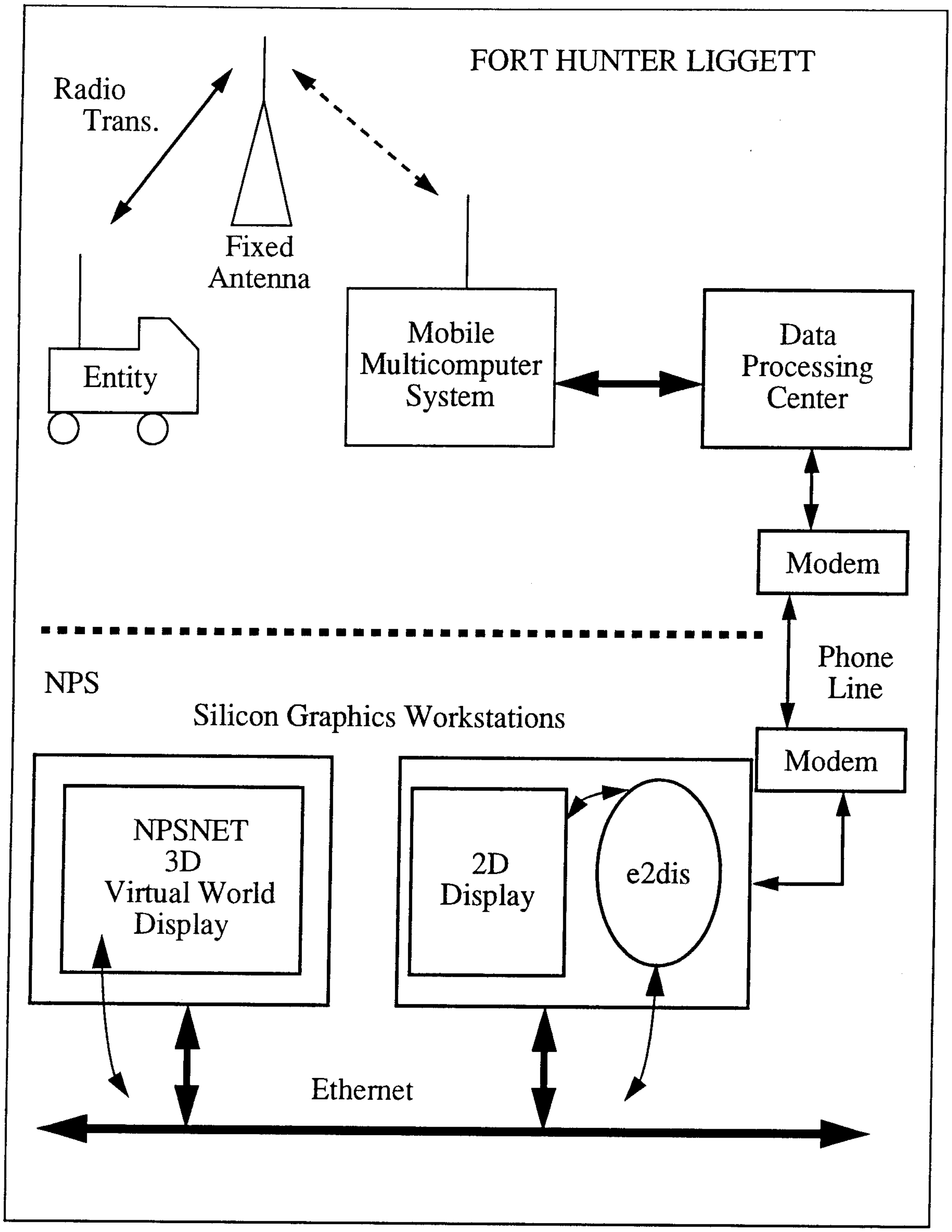


Figure 2. E2DIS Communications Configuration

4. DIS Integration into NPSNET

The work of Capt. Steven Zeswitz, USMC, [ZESW93] and John Locke, a Naval Postgraduate School computer specialist, in integrating DIS into NPSNET resulted in the development of a network library that provides network management and contains functions to open, configure, write to, and receive from an ethernet local area network using the DIS network protocols. As seen in Figure 3 below, the network library acts as an interface between the application and the operating system. This allowed NPSNET to be included in distributed simulations involving numerous types of simulators. This creation of a network library was useful in the development of the local area communications software architecture for the Janus-3D Visualizer.

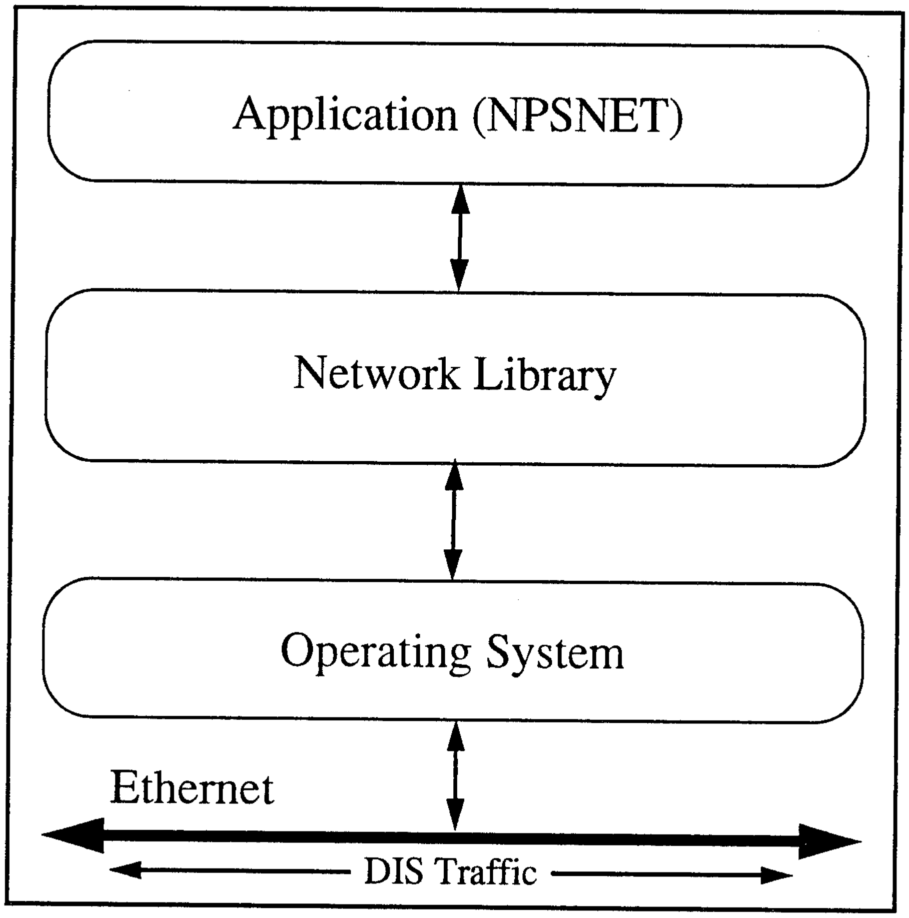


Figure 3. NPSNET DIS Network Interface

5. Summary

Much research has been conducted in the areas of both local and wide area networking of virtual environments and simulations. However, little has been conducted on their combined use in a single simulation. In this project, the concepts used in the earlier research were employed to combine both types of communications in order to extend a single local simulation over a wide area to enhance its overall training effectiveness.

III. JANUS-3D VISUALIZER OVERVIEW

A. BACKGROUND

As indicated in Chapter I, the Janus-3D Visualizer project is a result of the SIMITAR initiative. The purpose of the Visualizer is to improve the command, control and communication skills of National Guard battalion and brigade commanders and their staffs. It is hoped that this training will alleviate some of the weaknesses brought out during the National Guard's mobilization for Desert Shield/Desert Storm in 1990 and 1991.

B. VISUALIZER OVERVIEW

The Janus-3D Visualizer accomplishes its mission by showing a 2D battle running on the Janus Combat Modeler in three dimensions. This allows the commanders and their staffs a much better perspective of the battle. By viewing the battle in 3D, commanders get a better 'feel' for the terrain and its effects on movement and weapons' positioning. They will be able to use this view to critique their junior leaders' and staffs' performance both during and after the battle. Also, because the battalions and brigade will be networked, the brigade commander will be able to view the battle from his battalion commanders' viewpoints and critique overall brigade command and control.

C. EQUIPMENT

The equipment involved at each location includes that used for the Janus Combat Modeler and that for the Janus-3D Visualizer. The Combat Modeler setup at each battalion and brigade consists of a Hewlett-Packard Model 715/50 host computer linked to 20 Sun Sparcstations via an ethernet local area network. A three-button mouse also accompanies each workstation along with a 19-inch color monitor and keyboard.

The Janus-3D Visualizer equipment consists of a Silicon Graphics (SGI) Indigo 2 Extreme graphics workstation also connected to the Janus ethernet local area network. It also includes a three-button mouse, keyboard, 19-inch color monitor and a U.S. Robotics 28,800 BAUD modem linking the SGI workstation to the other SGI workstations at the battalions and brigade headquarters via commercial phone lines. This equipment was selected based upon its capabilities and as part of the dual use program as the workstations can be used for many other purposes at the National Guard armories when Janus battles are not being executed. Also, modems and commercial phone lines were selected for the wide area links as they met both minimum requirements and cost constraints.

D. SOFTWARE

The Janus-3D Visualizer is a set of programs that reside on the SGI Indigo 2 Extreme graphics workstation. The programs are written primarily in ANSI C with some C++. The files are broken down into three groups: terrain generation and data storage, graphics generation, and communications. A directory structure and file descriptions are shown in the User's Guide in the Appendix.

E. FUNCTIONALITY DESCRIPTION

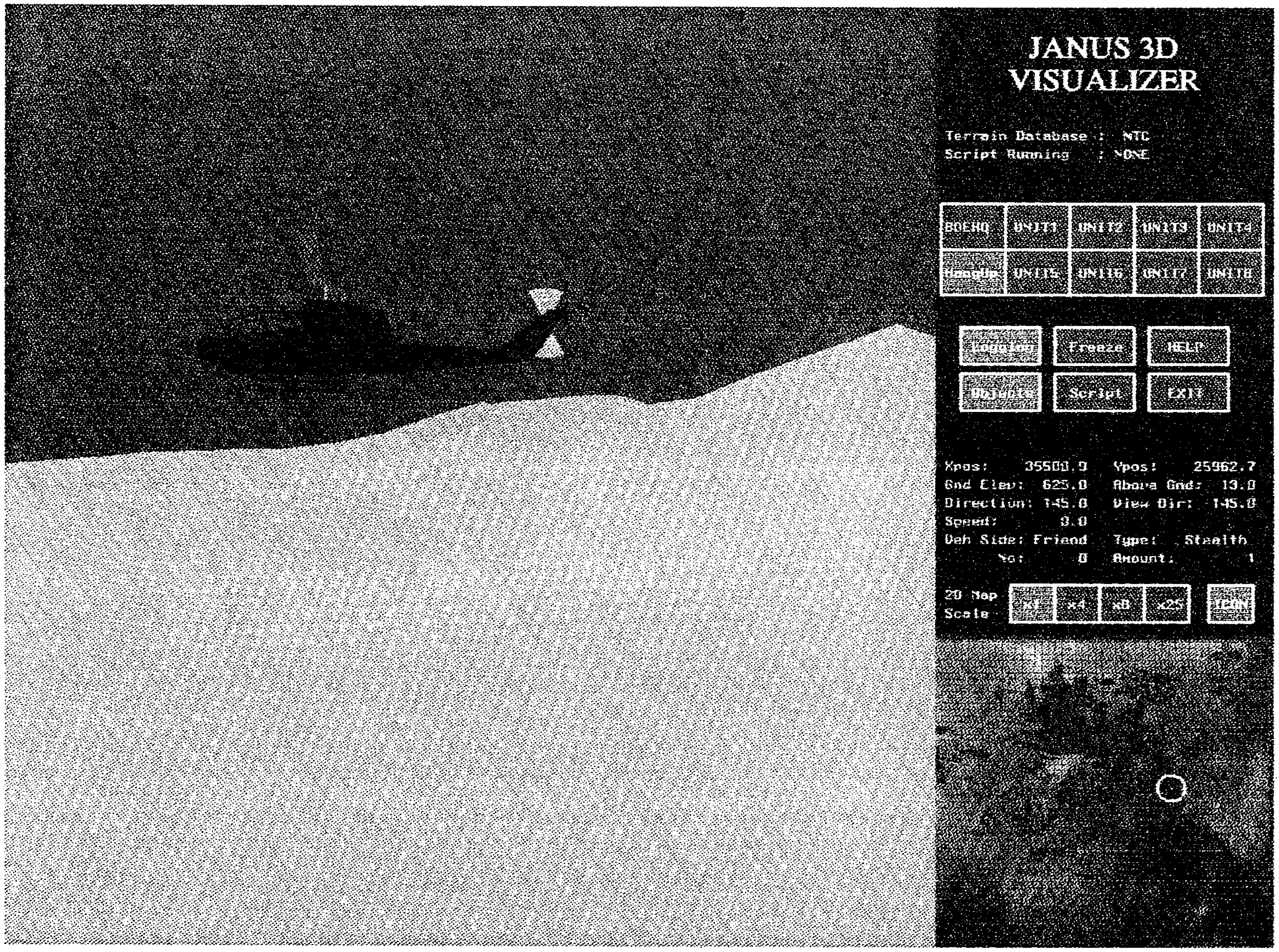
The Visualizer allows the user to view a currently running Janus battle in three dimensions via two different viewing methods. The first is tether mode. In this mode, the user can tether his viewpoint to a particular vehicle in the battle and see whatever that vehicle's field of view will allow as it moves around the battlefield. The other method is stealth mode. In stealth mode, users can simply fly around the battlefield traveling and looking wherever they want, as fast as they want. In other words, the user has a god's eye view of the battle. The tethered mode will allow leaders to better position their forces and to get a feel for the terrain as they see it from the vehicle's point of view. The stealth mode, on the other hand, will allow commanders to view the battle from anywhere they like in order to watch their junior leaders control their forces and to provide guidance and critiques both during and after the battle.

The Janus-3D Visualizer's main screen is shown in Figure 4. It is basically divided into three sections: the 3D viewport, the 2D map and the information and control panel. The 3D view depicts the battlefield in three dimensions as seen from a vehicle in tethered mode or from the god's eye view in stealth mode. Vehicles will be moving along the terrain in the 3D window in relation to there movements and positions in the Janus battle being executed on the HP workstation.

The 2D map shows the terrain in two dimensional gray scale elevation representation. Also, unit icons are depicted on the map in their positions relative to the Janus battle occurring on the HP workstation. The viewing triangle represents the vehicle's field of view that is depicted in the 3D window.

The information and control panel contains both pertinent information about what is currently occurring in the program and buttons allowing user interaction. The noninteractive information includes the terrain database being used, whether or not a script is being run, and stealth or tethered vehicle information.

The various interactive buttons allow the user to adjust the operations of the program. The 2D map scale buttons allow the user to change the 2D map scale in much the same way as can be done in Janus. The icon button is a toggle between numbers and symbols for the icons shown on the 2D map. Other functionality allowed by the control panel includes the creation and storage of script files of the battle with the 'logging' button. The 'freeze' button allows the user to stop vehicle movement while a script file is being run in order to allow the stealth vehicle to move around the frozen battlefield. Objects such as trees and urban areas can be displayed on the terrain using the 'objects' button. Also, the script button is used when the user wants to run a script file and view past action in the battle. Finally, the block of unit buttons is used to call and establish a connection with a remote unit. Once connection is established, Protocol Data Units (PDUs) from the remote unit will be transmitted and displayed on the local screen. The local unit can then see the remote unit's vehicles on their screen and observe their actions in the battle.



```
Terrain Database :  NTC
Script Running   :  NONE

Xpos:      35500.9   Ypos:     25962.7
Gnd Elev:  625.0     Above Gnd:  13.0
Direction: 145.0     View Dir:  145.0
Speed:       0.0
Veh Side: Friend     Type:   Stealth
      No:      0     Amount:       1

2D Map
Scale
```

Figure 4. Janus-3D Visualizer Main Screen

F. COMMUNICATIONS OVERVIEW

The focus of this research consisted of developing the software communication architecture for the Janus-3D Visualizer communications network depicted in Figure 5. The network consists of two main components. First, there is the ethernet local area network (LAN) that connects the Visualizer to the Janus Combat Modeler. Then, there is the wide area network (WAN) that connects the Visualizers of the different battalions and brigade headquarters to each other via modems and commercial phone lines.

The local area network consists of the HP workstation running the Janus scenario, 20 Sun Sparcstations depicting the Janus battle in its normal 3D representation and on which the users run the battle, and the SGI graphics workstation on which resides the Janus 3D Visualizer. Across this LAN, the HP transmits PDUs to the SGI. The SGI then uses the information in the PDUs to draw the battle in 3D on the same terrain database as is used in the Janus scenario. Unit movement, direct fire, indirect fire and detonation PDUs are sent across the network, interpreted by the Visualizer and then drawn. This all occurs in real time as the scenario is being run and allows the users to see in 3D what is happening in the local Janus battle.

The WAN, on the other hand, consists only of the SGI workstations at each battalion and the brigade headquarters. Each unit has the capability to call any other unit in the brigade, but only one at a time. The connection is established via modem and commercial telephone lines. Once a connection has been established, the same types of PDUs sent across the LAN are now transmitted from the remote SGI to the local or calling SGI. These PDUs are then interpreted and drawn concurrently with the locally generated PDUs. Now the local unit can see not only what is happening in its portion of the battle, but also what its sister unit is doing. This networked 3D viewing will allow for a much better perspective of the battle by unit commanders and their staffs

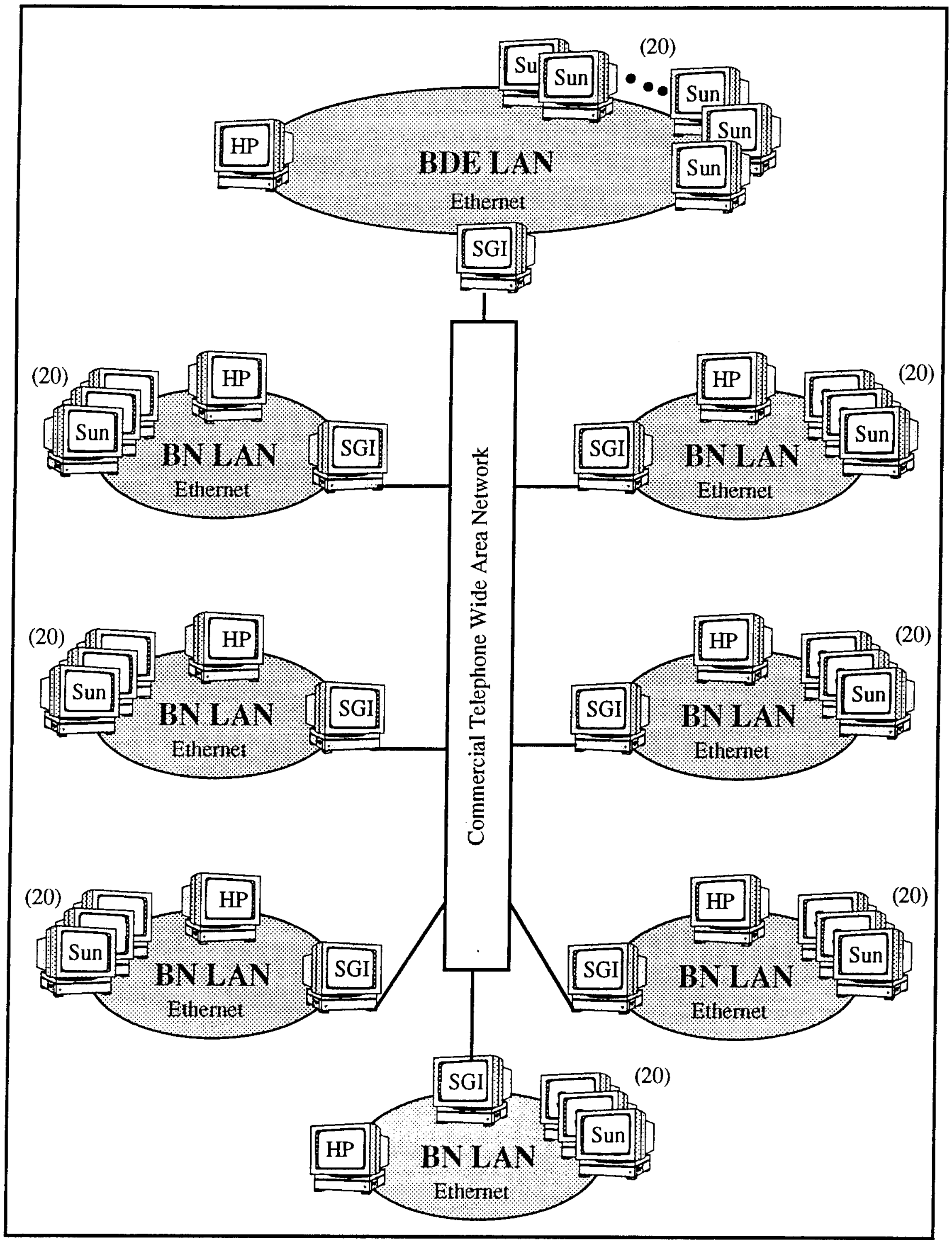


Figure 5. Janus-3D Visualizer Communications Network

IV. LOCAL AREA NETWORK MESSAGE TRANSFER

A. OVERVIEW

The LAN portion of the Janus-3D Visualizer's communications architecture is responsible for completing a number of missions. Its first task is to set up the communications on both the HP running Janus and the SGI running the Visualizer. Both the HP and the SGI contain a communications library that acts as an interface between the applications and the ethernet. It is these libraries that actually set up the connection with the network and place messages on and pull messages off of the ethernet. Once the network connection is set up, the remaining tasks occur after the simulation has started. All remaining tasks apply to how a PDU is handled from its creation on the HP to its display on the SGI. These operations, in the order they occur, are indicated below.

- HP: Extract necessary information from Janus.
- HP: Build PDU.
- HP: Transmit PDU to the Janus-3D Visualizer.
- SGI: Read PDU off of the ethernet.
- SGI: Write PDU to a script file.
- SGI: If connected to a remote unit, send PDU across telephone line.
- SGI: Store PDU in linked list.
- SGI: Read PDU from linked list for display by the Visualizer.

Each of these tasks will be discussed in this chapter except for remote connections, which will be detailed in Chapter V.

B. INFORMATION REQUIRED FROM JANUS

The information required for displaying a Janus scenario in three dimensions includes data on the entities to be displayed and their movements, direct and indirect fires, round impacts and kills. This information is retrieved from Janus by function calls placed into the Janus code itself that retrieve the required data. These calls are to C programming language functions that reside in the **janus/comm** directory in the **dis.c** and **send_fire.c** files. The amount of modifications made to the Janus source code in order to extract necessary information had to be kept to a minimum in order to prevent slowing of scenario execution. A breakdown of the Janus files that were modified and the types of modifications in terms of lines of code added are shown in Table 1. Also, in order to enable the Visualizer to keep up with the Janus Combat Modeler as a scenario is being executed, this information must be kept to a minimum to allow for quick transfer to and interpretation by the Visualizer. The following paragraphs describe the information needed and where it is found in Janus.

Janus Directory	File Name	Lines of Code Added	Declara-tions	Assign-ments	Function Calls	Other
MAIN	JANUS	2	0	0	2	0
INIT	INITMAIN	20	6	8	1	5
MAIN	WRITMOVE	16	6	9	1	0
ASSESS	SUSTN	9	3	5	1	0
ASSESS	IMPACT	5	1	3	1	0
DIRFIR	SHOOT	5	0	4	1	0
DIRFIR	ADFIRE	5	0	4	1	0
HELI	SFSHOOT	5	0	4	1	0
MOVE	UPDATE	16	6	8	1	1
MOVE	SETDLAY	16	6	8	1	1
ASSESS	DFMPACT	15	6	8	1	0
TOTALS	**11 Files**	**114**	**34**	**61**	**12**	**7**

Table 1. Janus Code Modifications

1. Unit Type and Graphics Symbol

In order to draw the proper two dimensional icon and three dimensional model in the 3D Visualizer, the Janus system type number must be retrieved. This information is included in all movement protocol data units (PDUs) and is cross-referenced against a copy of the system type table from the National Guard master data base that resides on the Visualizer[VAGL94]. In order to retrieve the system type, an index into the master system type table is retrieved from the scenario specific symbol table. This mapping is done wherever vehicle movement information is gathered: **initmain**, **writmove**, **update**, **setdlay** and **dfmpact**.

2. Initial Unit Locations

The initial locations of the units are obtained from **initmain** immediately after the initial scenario planning is saved by Janus. This information is necessary for drawing the units in the proper locations in the Visualizer prior to the start of scenario execution.

3. Unit Movement

Unit movement information is obtained from a number of files and can be broken down into units that are currently moving and those that have stopped. Information for all moving units is obtained in **writmove** as this routine is executed by Janus in order to write their movements to a file.

Unit halts, however, must be obtained from other sources as **writmove** is not executed when a unit stops. Three different Janus functions are used to determine when units stop: **update**, **setdlay** and **dfmpact**.

The purpose of the **update** routine is to process movements of both air and ground units. If a unit is moving, a call to **writmove** is made. However, if it stops, no such call is made. Therefore, a check was placed at the very end of **update** that tests to see if a unit's speed is zero. If it is, a call to **send_move** is made.

The zero velocity test in **update** does not catch all unit stops, however. It fails to see artillery vehicles that have stopped to shoot and foot soldiers that have been suppressed. These situations are caught as they occur in **setdlay** and **dfmpact**. In **setdlay**, which sets or cancels movement delays, a velocity check was placed at the end of the routine. Just as in **update**, if the unit's speed is zero, a call to **send_move** is executed. This test will catch those units that must stop to fire their weapons.

The **dfmpact** function processes direct fire impact events. One of the numerous tasks that it executes is suppression of foot soldiers, which results in their speed going to zero. At this point, a call to **send_move** is made in order to stop these units in the Visualizer.

4. Direct Fire Information

Information for direct fire events is obtained from the **shoot**, **adfire** and **sfshoot** functions. The **shoot** routine is used to simulate and evaluate direct fire events for normal units while the **sfshoot** function does the same for special and flyer units. Air defense weapon direct fire events are processed by **adfire**. In all three functions, a call to **send_fire_event** is made immediately after Janus writes the event to a recording file.

5. Indirect Fire Information

Indirect fire information is split into fires and impacts as these events do not occur virtually simultaneously as direct fire firing and impacts do. Indirect fire firing events are processed by the **sustn** function. This routine handles all indirect fire missions to include smoke, chemical, precision guided and artillery-delivered minefields. A call to **send_if_fire_event** is made immediately after the event is recorded to a file by Janus. Indirect fire impact events are processed by the **impact** routine. This function assesses the impact effects of all types of indirect fire munitions. The call to **send_if_impact_event** is made immediately after the impact event is written by Janus to the artillery file.

6. Unit Attrition Information

Within Janus, units can be aggregated or grouped together such that one icon may represent more than just one individual soldier or weapon system. For example, unit number 101 may be an individual rifleman icon on the screen. However, it may actually represent up to sixteen individual soldiers due to aggregation. Therefore, as that unit progresses in the Janus battle, kills are represented as decreases to the number of individual entities that an icon represents. This number is the NSCORE entry in the GLOBUNITS array. Once this number reaches zero, the unit is completely destroyed and its icon is removed from the screen.

Initially, we tracked these kills in the **kill** function with a call to **send_move**, which includes a variable for the aggregation level of a unit, whenever a unit was completely destroyed. It was noticed when examining the script files on the Visualizer, however, that whenever a unit was completely killed, duplicate messages were sent, one right after the other. One message was generated by **kill** and the other by **writmove**. Since **writmove** captures all unit kills, including those that do not reduce NSCORE to zero, the **send_move** call was eliminated from the **kill** routine.

C. MESSAGE FORMATION AND TRANSMISSION

There are four types of PDUs sent from Janus to the 3D Visualizer. They include the JanMovementPDU, JanDirFirePDU, JanIndirFirePDU and the JanDetonationPDU. The breakdown of the PDUs and their fields is shown in Figure 6. The PDUs are built and broadcast on the ethernet by the functions whose calls are embedded in the above mentioned Janus files.

1. Movement Message

The movement message, or JanMovementPDU, is built by the **send_move** function in the file **dis.c**. As mentioned previously, calls to **send_move** are made in the **initmain**, **writmove**, **update**, **setdlay** and **dfmpact** Janus functions. These calls cover all vehicle movements to include when they stop and start moving.

The information contained in the PDU is derived from that passed in to the function. Obviously, the type is used to determine what action to take upon the message's receipt in the Visualizer while the time stamp is included mainly as information that may be useful in reviewing script files.

The entity number indentifies the specific unit in the Janus scenario. Blue forces are represented by entity numbers 1 through 600, while the red forces are assigned 601 through 1200. Numbering the units in this way eliminates the need for a force identification field and thus reduces message size.

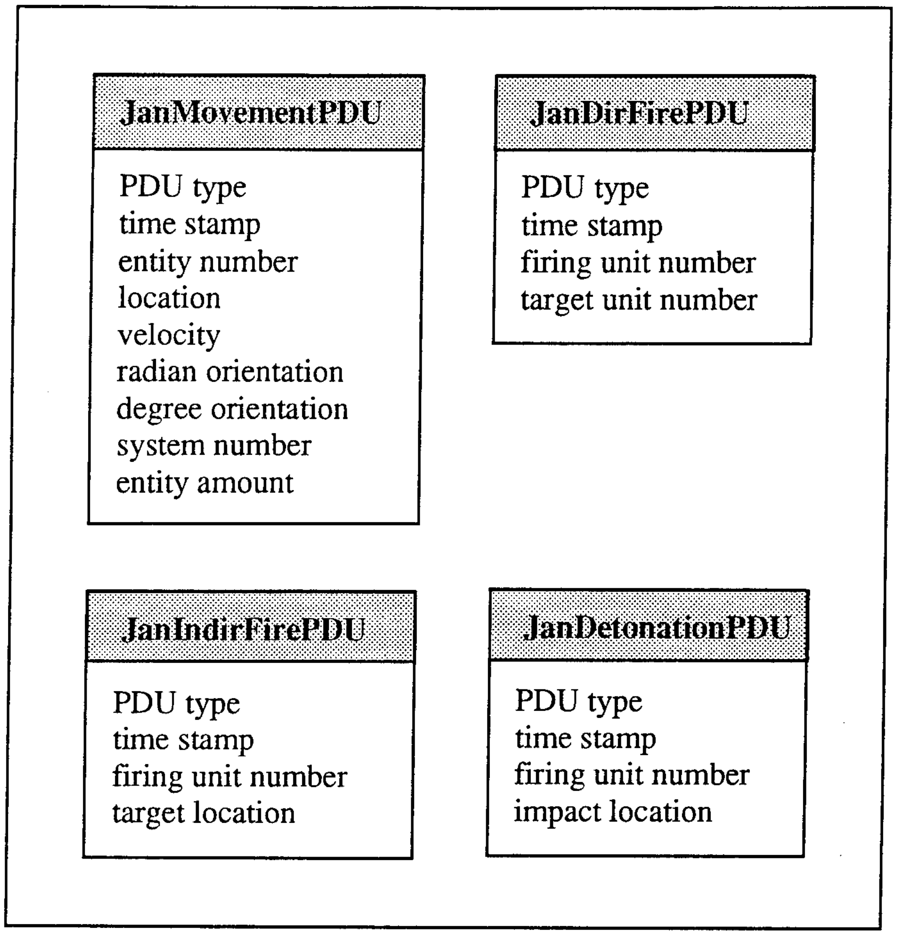


Figure 6. Janus-3D Visualizer PDU Contents

The entity amount field represents the value of NSCORE, or the number of vehicles or soldiers that this particular Janus unit represents. This number decreases as elements are killed during the Janus battle. When this number reaches zero, the entire unit has been killed. The 3D Visualizer monitors this field and when its amount reaches zero, the 2D map icon is drawn green and a dead 3D model is drawn in the 3D viewing area. In addition, this quantity is displayed on the information panel of the Visualizer whenever the user is tethered to a unit to let him know how many entities the icon currently represents.

The entity location field consists of the unit's x, y and z coordinates on the battlefield. Currently, the Janus x and y coordinates are converted such that the lower left corner of the map is considered the origin, (0,0), and kilometers are converted to meters for subsequent display in the Visualizer. Because Janus is a 2D simulation, it has no z value. Therefore, the z location field is filled with the distance the vehicle has traveled since its previous move.

The unit's velocity in kilometers per hour is contained in the entity velocity field. A unit's velocity is computed in **send_move** by determining the distance the unit has traveled since its last move and dividing it by the time delta since its last move.

Next, we have the entity orientations. Currently, the unit's orientation is computed in both radians and degrees. These values are computed based upon the unit's direction of movement, which is determined by the vector formed between the unit's previous and current locations. This Janus value is then converted into a value usable by the Visualizer. This conversion is necessary because, in Janus, zero degrees is East and rotation is counter-clockwise while in the Visualizer, North is at zero degrees and rotation is clockwise.

Finally, the system type number of the entity is included. As mentioned previously, it is this number that is mapped against the symbol table in the Visualizer to determine what 2D icon and 3D model are drawn.

With respect to movement messages, it was noticed when studying the script files that when a unit stopped, numerous updates were sent indicating that the unit was still stopped and nothing has changed. These messages are generally not needed by the Visualizer and serve only to increase the processing load it must work through. Only if a unit has suffered kills while stopped is a message necessary as the unit information panel on the Visualizer must be updated. Also, if the unit is completely destroyed, the icon and model on the Visualizer must be changed. Therefore, a test was inserted into **send_move** that filters out all subsequent updates of a unit who's velocity is zero except for those whose number of entities has changed. This results in a significant reduction in message traffic as these zero velocity updates occur at two to six second intervals for up to a minute and then at twenty second intervals thereafter until the vehicle begins to move again or is completely destroyed. Also, in large scenarios, it is likely that 50% or more of the units are stopped at any given time.

2. Direct Fire Message

The JanDirFirePDU is constructed in the **send_fire_event** procedure residing in **send_fire.c**. This procedure is called from within the **shoot, sfshoot** and **adfire** functions of Janus in order to display direct fire engagements in the 3D Visualizer. These calls cover all direct fire engagements.

The direct fire message is very simple. It contains only the firing and target unit numbers along with the PDU type and time fields. The unit numbers are used to display direct fire lines from the shooter to the target in the Visualizer's 3D window[VAGL94].

3. Indirect Fire Message

The indirect fire message, or JanIndirFirePDU, is built in the **send_if_fire_event** procedure, also within **send_fire.c**. Calls to this procedure are made from **sustn** and are used to display muzzle flashes on the firing vehicles in the 3D Visualizer.

Like the direct fire message, this PDU is also simple. It contains the message type and time stamp as all messages do. It also contains the firing entity number and target location. The firing entity number is used to determine the vehicle to apply the muzzle flash to while the target location is used to determine the barrel or turret orientation [VAGL94].

4. Detonation Message

The last PDU type is the JanDetonationPDU. This message is constructed in **send_if_impact event** in **send_fire.c**. This procedure is called from within **impact** and is used to display indirect fire impacts in both the 2D map and 3D window displays on the 3D Visualizer.

This message includes the message type, time stamp, firing entity number and impact location. The impact location is used to determine where to draw the impacts in the Visualizer while the firing entity number is currently included to be used for debugging purposes.

5. Message Transmission

Within each of the four functions that build the PDUs, a call to **net_write** is made that actually transmits the PDU across the ethernet. This **net_write** function is part of the communication library developed by John Locke and modified for this project, that resides in the **comm/DIS-2.0** directory. As mentioned previously, this library also initializes the connection with the ethernet during Janus' initialization process with a call to **net_open**.

D. MESSAGE HANDLING IN THE JANUS-3D VISUALIZER

PDUs received from Janus across the ethernet LAN are handled by the communications library in the Janus-3D Visualizer. These messages serve three purposes. The primary purpose, of course, is concurrent 3D display of the Janus battle. Also, the messages are stored in script files to be used for later after action reviews. Finally, if a telephone call is received from another unit, the messages are sent to that unit via the telephone connection for display in the remote unit's Visualizer. The first two purposes will be discussed below while the telephone transfer is described in Chapter V.

1. Ethernet Communications Library

The ethernet communications library, initially developed by Capt. Steven Zeswitz and John Locke and modified for this project, resides in the **visualizer/src** directory [ZESW93]. The ethernet connection is established during the initialization process of the 3D Visualizer with a call to **net_open**, which includes the ethernet interface. Besides setting up the communications sockets, this routine also spawns a receiver process, **receiveprocess**, that runs concurrently with the Visualizer main process and reads traffic off of the ethernet. Finally, an arena, or shared memory area, is set up in which message information and a number of communication related variables and flags are maintained.

The arena is shared by the Visualizer application process, the ethernet receiver process and the modem communications processes. This allows for much more efficient data handling as all user processes have access. As mentioned above, this area of shared memory contains a number of variables and flags that are used by the various processes to determine what tasks are to be executed. Those that apply to local display of the Janus battle include pointers to the head and tail of the linked list of PDUs, a counter for the number of PDUs in the list, and a pointer to the semaphore that prevents simultaneous access of the linked list by two or more user processes.

2. Script File Creation

Script files are created for the Janus scenario during its execution. The purpose of these script files is to act as a tool for unit leaders to use to enhance after action reviews of the Janus battles. With these files, the scenario can be replayed in its entirety after the battle has been completed. Commanders can go over key actions during the battle and point out both good and the bad execution. Any script file can be selected for viewing and motion can be stopped or frozen during the review. This allows the leaders to explore the battlefield in the stealth mode while all is still. [VAGL94]

The script files are created in the **receiveprocess** of the communications library. The first task is to build the file name. This name is built such that the script files are stored in a directory based on the terrain that is being used for the scenario. For example, if the terrain being used is NTC (the National Training Center at Fort Irwin, California), the directory in which the script files for that scenario will be stored is **visualizer/terrain/NTC/scriptfiles**. Within that directory, the individual file names are **datafile** followed by their sequence number. For example, **datafile.0005** would be the fifth script file created for the scenario.

After a PDU is read off of the ethernet and verified to be one of the four valid PDU types, it is written to the script file by a function called **printPDU** that is located in **src/print.c**. This routine actually prints the PDU to the file in ASCII format. The ASCII format allows the user to read the files if necessary and for easier debugging of the code. If it is found that ASCII script files use too much system memory, the routine can be rewritten to write binary files to save space.

Each script file remains open for twenty minutes based upon the time stamp of the incoming PDUs. The initial start time is set equal to the time stamp of the first PDU received while the stop time is simply the start time plus the time interval in seconds, or 1200 for a twenty minute file. After each PDU is written to the file, the value of the stop time is compared to the time stamp of the PDU. If they are equal, the current script file is closed, the new start time is set to the old stop time and the new stop time is set to the new start time plus the time interval. A new file is then opened with an extension one greater than the previous file. This process continues until the Janus scenario has ended and the 3D Visualizer has been terminated.

Because any script file can be opened first and not just the initial one, each script file must begin with the current status of each unit. This is necessary because after the Janus simulation has started, JanMovementPDUs are rarely generated for stopped units. As a result, not all units would be included in the script file and therefore placed on the Visualizer battlefield if they did not move or suffer a casualty during the file's time interval. This would decrease significantly from the realism and accuracy of the battle replay. To alleviate this problem, an array is maintained by the **receiveprocess** that contains the information from the last JanMovementPDU received for each unit. In other words, each time a JanMovementPDU is received, its contents are stored in this array named **position_array**. Whenever a new script file is opened, each entry in **position_array** is written to the script file in the exact same format as a JanMovementPDU. As a result, each script file begins with every unit's status as of the end of the last script file. Therefore, no matter which script file is selected, all forces will be displayed on the battlefield in their correct state. The initial script file obviously does not require this as it begins with the initial position dump received from Janus just prior to the battle's start.

3. PDU Storage In Linked List

The next action that takes place within the LAN portion of the communications architecture is the storage of the PDUs into the linked list in the arena. This is a simple process. First, the linked list semaphore is obtained. The PDU is then added to the tail of the list and the tail pointer variable maintained in the flags and variables structure in the arena is updated. Of course, if the list is empty, the PDU is added to the head of the list and both the head and tail pointer variables are updated. This linked list grows with each PDU received off of the ethernet and shrinks with each PDU read by the Visualizer for display.

4. Reading the PDUs for Display

The whole purpose of the Janus-3D Visualizer is to display the Janus battle in three dimensions. This is accomplished by reading each PDU from the linked list in the arena and using its information to display it in both the 3D display and 2D map. The functions that retrieve the PDUs and update the unit status arrays in the Visualizer are contained in **visualizer/network.c**.

The display process is started in the main application loop of the Visualizer, in the file **jeep.c**, by a call to the **getpackets** function, which resides in **network.c**. The getpackets routine then calls **net_read**, which resides in the communications library. The **net_read** function actually reads and returns the PDU at the head of the list and then updates the head of the list pointer maintained in the shared memory variable list. The **getpackets** function then determines what type of PDU has been received and, based on that type, determines the next action to be taken. This function continues until all of the PDUs in the linked list have been read. After the last one has been read, control is returned to the Visualizer's main application loop in order for the drawing routines to be executed.

As mentioned above, the action taken within **getpackets** depends upon the type of message received. If the message is a JanMovementPDU, a function called **getjanusmovemess** is executed. This routine simply takes the information from the movement PDU and loads it into the vehicle status arrays of the Visualizer. All data except the system number is read into the **veharray** while the system type is read into either the **friendyvehtypearray** or **enemyvehtypearray** depending on which side it is on. It is these arrays that are read by the Visualizer's drawing routines to actually display the units on both the 2D map and in the 3D viewing window [VAGL94].

The remaining message types are acted upon by functions that lead to the drawing of direct and indirect fire effects on the Visualizer. If the PDU received is a JanDirFirePDU, the **getjanusfiremess** function is called. This function loads the shooter and target unit numbers into the **shotarray**. This information is then used by the Visualizer to draw the red and blue direct fire lines in the 3D window from the shooter to the target [VAGL94].

For indirect fire firing and detonation messages, the functions **getjanusindirfiremess** and **getjanusdetmess** are executed. If the message is the JanIndirFirePDU, the **getjanusindirfiremess** function is called by **getpackets**. This function calculates the gun orientation of the shooter based upon the location of the target in relation to the shooter. This information is then used to orient the unit's weapon toward the target and to display a muzzle flash [VAGL94]. When an impact, or JanDetonationPDU, is received, the **getjanusdetmess** routine is executed. This function uses the impact location to determine where on the 2D map and the 3D window to draw the explosion symbols. The **logexplo** function is then called to draw the explosions [VAGL94].

V. WIDE AREA NETWORK MESSAGE TRANSFER

A. OVERVIEW

The WAN segment of the Janus-3D Visualizer communications architecture is responsible for all modem operations. These operations include initialization, listening for connections, sending PDUs to and receiving PDUs from a remote unit, and displaying those PDUs in the Visualizer. The majority of these tasks require subprocesses, specifically the listening and receiving processes, to be spawned. These WAN functions are found primarily in **visualizer/modem.c**. A conceptual view of how the modem operations are conducted is shown in the flow chart in Figure 7.

B. INITIAL MODEM CONFIGURATION

Prior to using the Janus-3D Visualizer, the modem must be connected to the SGI. As currently written, port 2 is the serial port opened for the modem. Changing the port is simply a matter of changing the MODEM_PORT definition in **modem.h**, which is in the **visualizer/headers** directory.

The modem used in this project was the USRobotics Sportster 9600 high speed, full duplex data modem with V.32 transmission. For the Visualizer's operations, the DIP switches were set as follows:

• 1: UP - Normal Data Terminal Ready (DTR) operations.

• 2: UP - Verbal result codes (vice numbered).

• 3: DOWN - Result code display enabled.

• 4: DOWN - Command mode local echo suppressed.

• 5: UP - Auto answer enabled for pick up on first ring.

• 6: UP - Modem sends carrier detect (CD) signal upon connection and drops CD on disconnection.

• 7: DOWN - Load factory settings from read only memory (ROM).

• 8: DOWN - AT (attention) command set recognition enabled.

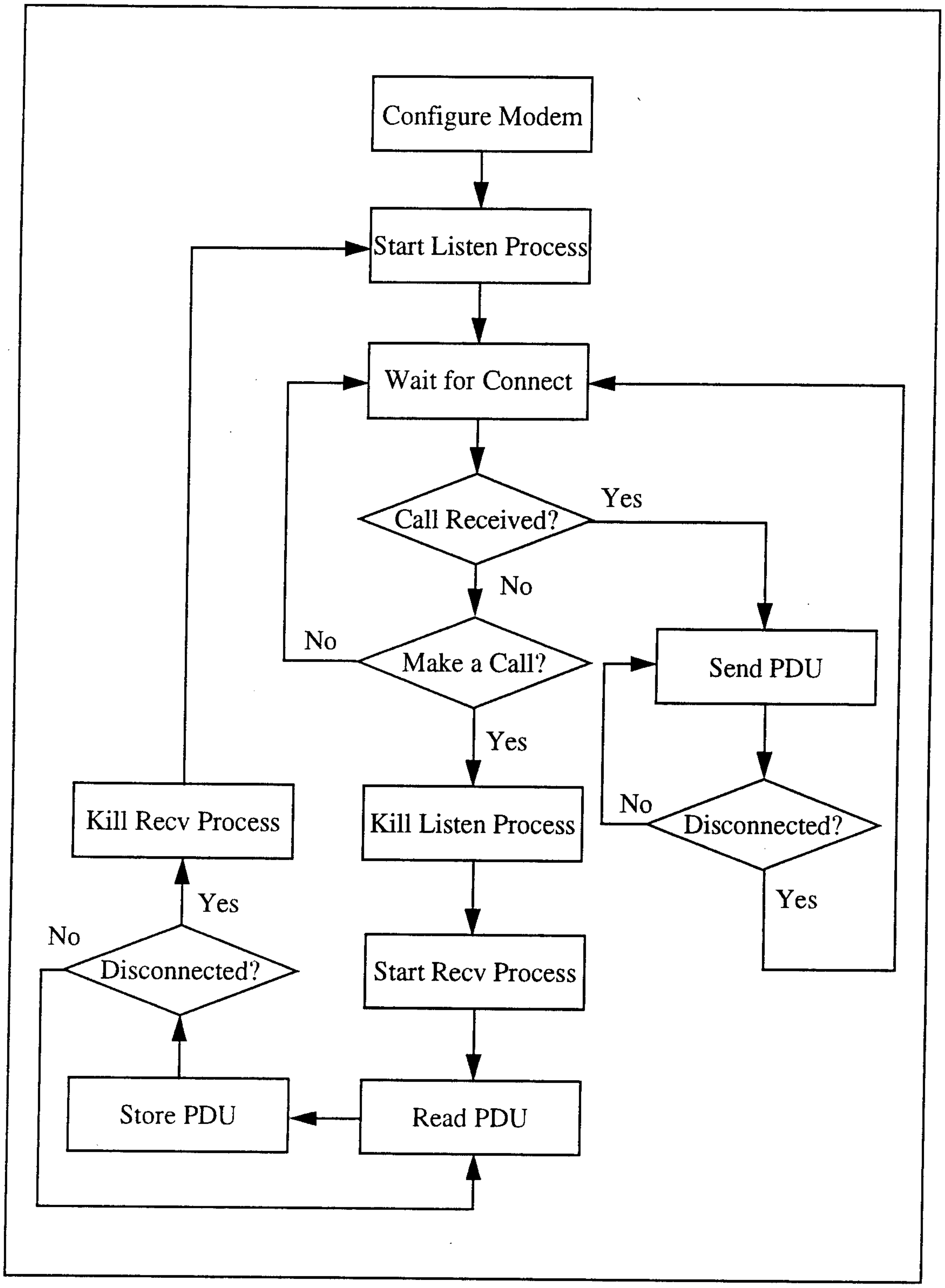


Figure 7. Conceptual View of WAN Operations

C. STARTING THE LISTENING SUBPROCESS

During the initialization of the 3D Visualizer, a call to **modem_open** is made just prior to the start of the main application loop. The **modem_open** function resides in **modem.c** and its sole purpose is to call the **start_listen_process** function. This function then spawns a child process called **Janus_write_process** that runs concurrently with the main application process and the ethernet **receiveprocess** mentioned in Chapter IV.

It is the **Janus_write_process** that actually initializes the modem by sending it initialization commands. First, the ATZ command is issued to reset the modem and ensure it is operating according to the DIP switch settings. Next, a set of default settings are issued as an extra precaution against the DIP switches being wrong. These settings are: [USRO92]

• &A0 - ARQ (error control) codes are disabled.

• &B0 - Variable rate at the serial port interface which allows the modem to change this rate to match the connection rate established with the remote modem.

• &C1 - Normal CD operations.

• &I0 - Disable XON/XOFF flow control of received data.

• &N0 - Variable phone line interface rate which allows the modem to negotiate the connection rate with the remote modem.

After the default settings are passed in, another set of operating parameters is sent. These include disabling local echo, displaying result codes and displaying them in text, keeping the modem speaker on until a connection is established, and answer on one ring. Again, these are passed in to ensure the settings we want are used.

After this initialization takes place, the modem simply listens for an incoming telephone call from a remote unit. If a call comes in, it will answer on the first ring as determined by the settings made during the initialization process. Because the ***Janus_write_process*** is a spawned subprocess, it does not hinder the execution of the 3D Visualizer while it listens for a call.

D. SENDING PDUS TO A REMOTE UNIT

1. Connection Establishment

If a call comes in from a remote unit while the modem is listening, that call will be answered on the first ring. Once the connection is established, two flags are set. These flags are the SEND_PDUs and JUST_CONNECTED flags that reside in the arena in order for them to be seen by the application process, and more importantly in this case, by the **receiveprocess** in the communications library.

2. Sending the PDUs

In the **receiveprocess** that continuously reads PDUs off of the ethernet and stores them in both a linked list and script file, the SEND_PDUs flag is checked immediately after a PDU is written to the script file. If the flag has been set due to a connection being made with a calling unit, the PDU is sent to that unit through the modem port.

However, if the JUST_CONNECTED flag is also set, which it will be immediately after the connection is established as mentioned above, the **position_array** discussed in Chapter IV comes back into play. When the JUST_CONNECTED flag is set, a JanMovementPDU is constructed for each unit in the array and sent to the remote caller. Just as in the beginning of each script file, the status of every unit at the time the telephone connection is made must be sent to the remote caller in order to ensure that all of the local units show up on the remote unit's Visualizer. As previously discussed, if this was not accomplished, only moving units would show up on the remote Visualizer, thus significantly impairing the ability of the remote users to determine exactly how the battle is progressing at the local unit. No stationary living or destroyed units would be displayed unless one happened to suffer casualties while the transmission was taking place.

After this initial download of the **position_array** information across the WAN, the JUST_CONNECTED flag is reset to zero and the array values are no longer sent. At this point, since the remote unit has received the position of all vehicles, only the current PDUs being read off of the ethernet are transmitted.

The **receiveprocess** continues to write PDUs to the modem port as long as the SEND_PDUs flag is set. Once it notices that the SEND_PDUs flag has been reset to zero, however, it stops sending PDUs across the WAN connection and simply continues to store the PDUs in the linked list and script files as it was doing before the connection was established.

3. Connection Termination

Upon the termination of the connection by either the remote caller or local sender, a number of actions take place on the sender's end. First of all, the SEND_PDUs flag is reset to zero. As mentioned above, this terminates the writing of the PDUs to the modem port in the **receiveprocess**. The 'hangup' command is then sent to the modem to ensure that the connection has terminated properly. This is followed by the closing of the file descriptor and the killing of the current **Janus_write_process**. Finally, the **start_listen_process** is called again to create a new **Janus_write_process** and reinitialize the modem. All of these actions take place in either the **hang_up** or **remote_hang_up** functions in **modem.c** depending upon who terminated the connection.

E. RECEIVING PDUS FROM A REMOTE UNIT

1. Connection Establishment

When a battalion wants to call one of its sister units to see what is happening in their part of the Janus battle, the user selects the call button of the unit to be called. This results in a call to **call_remote_unit** which is located in **modem.c**.

The **call_remote_unit** function first kills the **Janus_write_process**. This must be done in order to set up the receiving process. It then calls the **start_receive_process** function.

The **start_receive_process** accomplishes its mission by spawning the **Janus_receive_process.** This process then passes the port and phone numbers to **call_SGI**. It is this routine that actually initializes the modem just as in **Janus_write_process**, and then executes the call. Once the connection is established, a call to **readPDU** is made from within **call_SGI**. This call is executed continuously until the connection is terminated.

2. Reading the PDUs

The **readPDU** function in **modem.c** actually reads the PDUs sent by the remote unit. The information received is read one byte at a time. Once the first byte is read, the PDU type is determined. From this, the message length is derived and therefore the number of bytes remaining in the message is known. After a complete PDU is received, it is placed into the linked list for modem PDUs and the head and tail pointers to that list are updated after the modem list semaphore has been acquired.

3. Connection Termination

As previously mentioned, the **readPDU** call is made until the connection is terminated by either party. Again, depending upon who terminated the connection, either **hang_up** or **remote_hang_up** is called. The procedures within the hang_up functions are the same except that the process being killed on the receiving SGI is the **Janus_receive_process**. After this process is killed, another **Janus_write_process** is started and the modem sits and waits for someone to call or for a command to call someone else.

F. DISPLAYING PDUS RECEIVED VIA THE TELEPHONE WAN

PDUs received from a remote unit across the telephone network are in the same format as those received from the ethernet. As a result, the display procedures for these PDUs are virtually the same as those for the locally received PDUs.

The modem PDU display process is started much the same as the ethernet PDU display process. A call is made to the **getmodpackets** function immediately after the **getpackets** call in the main loop of the Visualizer. This function also resides in **network.c**. Similar to **getpackets**, **getmodpackets** calls a function to read and return a PDU from the modem linked list. In this case, the reading function is called **readmodpackets**, which operates very similar to the **net_read** function in the communications library. Once the

PDU is returned from **readmodpackets**, its type is determined and a call is made to the appropriate array updating function that will update the Visualizer's display arrays in order for the information in the PDUs to be displayed. The **getmodpackets** function will be called as long as a connection is maintained with another unit. Once the connection is broken, the **getmodpackets** function call is no longer made.

VI. PERFORMANCE, CONCLUSIONS AND FUTURE RESEARCH

A. PERFORMANCE

It was noted during this research that achieving a real time traffic flow over the ethernet local area network was not a problem. The 3D Visualizer residing on the SGI Indigo 2 Extreme had no trouble keeping up with the messages generated by the Janus battle and displaying them such that time and space coherence was maintained.

Using the USRobotics Sportster 9600 modem, however, resulted in a noticeable lag between the units in Janus and their counterparts on the Visualizer while the Visualizer was connected and sending PDUs to a remote unit. This delay was caused by the relative slow speed of the modem. It could not send the PDUs as quickly as they were being fed to it by the ethernet **receiveprocess**. This was to be expected as a 9600 BAUD modem can handle only 21 JanMovementPDUs (the longest PDUs used by the Visualizer at 56 bytes) per second. This equates to 48 movement updates every two seconds, the normal Janus update cycle length. However, with the 28,800 BAUD modems that the National Guard will be using with the 3D Visualizer, the number of movement updates that can be handled every two seconds is nearly 130. This will allow for real time transmission via the wide area telephone network as it is unlikely in a battalion sized scenario that 130 vehicles will be moving at the same time. Also, Janus does not update all moving vehicles every two seconds. The slower moving vehicles may have updates up to eight seconds apart, thus reducing prospective message traffic further.

B. CONCLUSIONS

The primary objective of this research was to develop a software communication architecture that would allow events occurring in two dimensions in the Janus Combat Modeler to be seen in three dimensions on the Janus-3D Visualizer over both local ethernet and wide area telephonic networks. This objective was accomplished. A Janus scenario can now be viewed in three dimensions as it is executing, thus enabling commanders and their staffs to view the battlefield more realistically and thus improve their training. Also, due to the wide area network capability, separate units can remain at their home stations and still participate in the same Janus scenario. This significantly reduces lost training time and funds due to extensive travel. As a result of this research, the Janus-3D Visualizer is currently being tested by the National Guard.

C. TOPICS FOR FUTURE RESEARCH

There are several areas or topics of future study that can be pursued as a result of or to improve upon this project with respect to communications. First of all, a set of monitoring tools for modem communications could be developed to assist in analyzing the performance of the WAN message transmissions. Such a set of tools should be able to analyze throughput and error rates just as similar tool kits do for local area ethernet networks, for example.

Another area of research that would enhance the effectiveness of the Janus-3D Visualizer would be to make it an interactive station. In other words, implement two-way communications between Janus and the Visualizer. This would allow the commanders and their staffs to take advantage of the more realistic three dimensional view provided by the Visualizer to employ their forces both prior to and during the battle, thus enhancing training effectiveness.

Finally, with the immense growth in popularity of distributed interactive simulations, their spread to a more widely varied group of users is sure to increase. With this spread comes the movement of these simulations to networks consisting mainly of phone lines as this is the least costly method of communications. As a result, there may well be a need to develop a standard low bandwidth telephonic communications protocol that would allow these simulations to be conducted over traditional telephone networks in real time.

APPENDIX: JANUS-3D VISUALIZER USER'S HANDBOOK

JANUS 3D VISUALIZER

(USER's HANDBOOK)

Naval Postgraduate School

Advisor: Dr. David Pratt

Written by: Chris Upson and Jim Vaglia

Directory Structure and Program Files

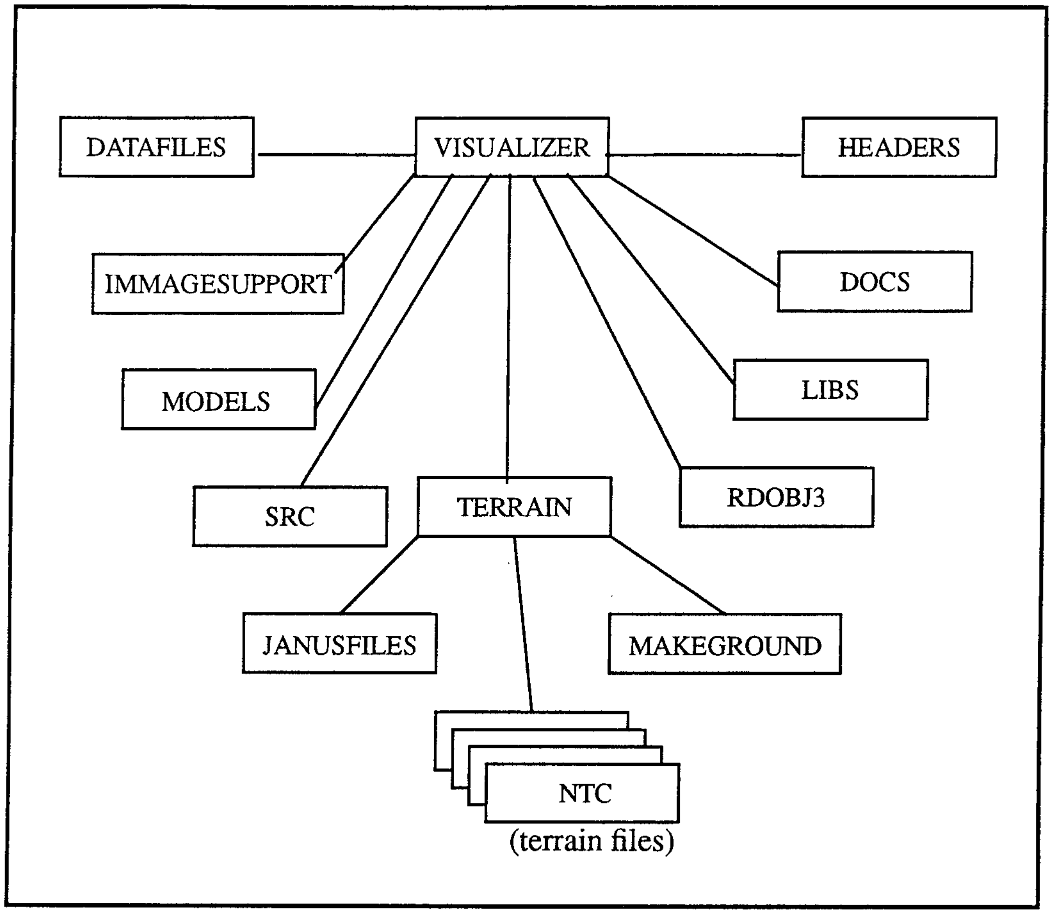


Figure A-1. Directory Structure

The program is broken down into the following directories and files.

Figure A-1 shows the directory structure of the Janus-3D Visualizer.

visualizer	The main program files are located here.
datafiles	This directory contains the input files.
headers	The header files for all the programs are located in this directory.

imagesupport	This directory contains the imaging programs.
libs	The header files for the manipulation of 'off' objects.
models	The three dimensional models are located in this directory.
rdobj3	The files to manipulate the 'off' objects are in this directory.
src	The communication programs are in this directory.
terrain	The terrain files, script, and terrain conversion programs are located in this directory.
docs	Contains Frame Maker postscript and text versions of the user's handbook and briefing slides.

The following programs are located in the visualizer directory.

acds.font	This contains the fonts used to produce the icons for the vehicles on the two dimensional map.
checkintersect.c	This file contains the routines for terrain intersections.
cover.c	This file contains the routines to open terrain files.
dogsncats.c	This file contains the functions used to switch from one vehicle to another and define the cursor.
drawcover3d.c	This file contains the routines to draw the 3d terrain.
drawobjs.c	This file contains the routines to draw the 3d objects.
fontdef.c	This file is used to manipulate the fonts.
infopannel.c	This file contains the display routines for the information panel
janus3d	This is the executable code for the visualizer.
jeep.c	This is the main program. It creates the display window and ties all of the other programs together.
jeepmot.c	This contains the procedures to move the vehicles.

map.c	This draws the two dimensional map and the icons that are displayed on it.
menus.c	This file contains the information that is displayed when the popup menus are selected.
modem.c	This file contains the routines for modem communications.
network.c	This file contains the routines to read the communication packets.
readfiles.c	This file contains the procedures to read in the terrain and vehicle data files.
readobjs.c	This file contains the routines to read the objects.
util.c	This file contains functions to fill in the polygons and place vehicles on the ground.
viewbounds.c	This file contains the procedures to display the three dimensional objects and terrain on the screen.

Installation Procedures

To install the Janus-3d Visualizer, load the file **visualizer.tar.Z** in the directory you want to install the program. Once loaded, uncompress the file by typing **uncompress visualizer.tar.Z [enter]**. When this is finished, untar the files. This will separate the program into the individual files and subdirectories. The command to do this is **tar -xvof visualizer.tar[enter]**. You are now ready to run the program in the default mode.

Communications Setup

1. Ethernet Network Setup

The interface with the local area ethernet network is set up and maintained by the network library in the **visualizer/src** directory. The **net_open** call just prior to the beginning of the main application loop in **jeep.c** establishes this interface. Ensure the value of **BCAST_INTERF** as defined in **headers/jeep.h** is correct for your network. You can check for your system's network interfaces by using the "netstat -rn" command. Also check the port definitions for the send and receive ports that are listed as **UDP_SEND_PORT** and **UDP_RECV_PORT**. If you change any definitions in **jeep.h**, ensure you recompile both the **src** and **visualizer** directories.

2. Modem Setup

These modem setup procedures apply to the US Robotics Sportster 9600 modem that was used in the design of the Janus-3D Visualizer prototype. If you have a different model modem, please consult its user's guide where appropriate to ensure these procedures will result in proper setup.

All modem processes, to include opening, transferring data and closing, are contained in the file **modem.c** in the **visualizer** directory. Its header file, **modem.h** is in the **headers** directory. The modem interface is established with the **modem_open** call immediately following **net_open** in the Visualizer initialization process.

Once you have connected your modem to the system, determine what serial port it is connected to. Then check this with the **MODEM_PORT** definition in **modem.h**. Our default is port "2". If yours is different, change the **MODEM_PORT** definition and recompile the **visualizer** directory. Also, even though default settings are loaded into the modem upon its initialization within the Visualizer, check the dip switches on the back of the modem to ensure that 1, 2, 5 and 6 are up and 3, 4, 7 and 8 are down. Pages B-4 and B-5 in the user's guide contain more detailed informations on the dip switch settings. Finally, once the Visualizer has completed its initialization process and is ready to run, the Auto Answer (AA), Data Terminal Ready (DTR) and Clear to Send (CS) lights on the front panel should be illuminated.

Terrain Conversion

To convert the Janus TERRAINxxx.DAT into files that are readable by the Visualizer follow these instructions. There are seventeen steps in the process. These steps need to be executed in the order presented. The conversion process takes awhile, suggest you run the programs running in the back ground. To run a program in the background type & after the command and prior the pressing enter.

The janus TERAINxxx.DAT needs to be placed in the janusfiles directory. You need to ensure that the temporary storage directories in the terrain directory are empty, some of the conversion files append to existing files, this will cause erroneous data to be stored in the files. Then execute the following steps to convert the terrain:

1. readtrrn <terrain #><terrain name>

This program reads the TERAINxxx.DAT located in the janusfiles directory.First the program uses the terrain name to create the root directory for the header files, terrain files and script files needed in the conversion process and the Janus-3D visualizer. The subdirectories created in the terrain directory are: elevfiles, objectfiles, quadfiles, scriptfiles and textobjectfiles. Readtrrn creates five files. The files globals.dat and janus.text are placed in the terrain directory. Globals.dat contains the map parameters and is used by the other conversion programs and Janus-3D Visualizer to initiate the global variables. Janus.text contains the same information but with the text names of the variables. The other three files are placed in the janusfiles directory. The file xxx.ele contains the map elevation and grid information. The remaining files; xxx.riv and xxx.road, contain the coordinates of the rivers and roads respectively.

2. gen_binary_elev <terrain #><terrain name>

The program reads in globals.dat and xxx.ele. Next the program creates the file elev.bin.dat which contains only elevation data and places them in the sub-terrain directory elevfiles.

3. make_tri_mesh <terrain #><terrain name>

The program reads in globals.dat and the elev.bin.dat file that was created in step three. Elev.mesh.bin is created containing the terrain mesh information and is placed in the same directory as elev.bin.dat.

4. conv_elev2block_bin <terrain #><terrain name>

Conv_elev2block_bin reads in globals.dat and elev.bin.dat and creates one kilometer grid square files. For a 50 Km by 50 Km map the program creates 2500 files and stores them in the elevfiles directory.

5. janus2nps <terrain #><terrain name>

Janus2nps reads globals.dat and xxx.ele files. The program then creates and places the file cover.dat in directory textcoverfiles. Cover.dat contains the elevation, normal, and colors of the points.

6. reverseroads <terrain #><terrain name>

Janus reads the map information from the lower left hand corner. NPSNET bases the location of objects on the upper left hand corner. This program modifies the coordinates of xxx.riv and xxx.road so they can be read into NPSNET terrain. The location of the files is in the roadrivfiles.

7. makeroadfile <terrain #><terrain name>

Makeroadfile reads the globals.dat, xxx.road and xxx.riv files. The program then creates the file roads.dat. This file contains the information and points needed to draw the rivers and roads as polygons.

8. makeroads <terrain #><terrain name>

Makeroads reads globals.dat and xxx.ele files. The program then creates and places the file roadcover.dat in directory roadrivfiles. Roadcover.dat contains the elevation, normal, and colors of the points.

9. makenewtrees <terrain #><terrain name>

This program extracts the density, city or tree, road, and river information from the xxx.ele file. The files treecover.dat and citycover.dat are created and stored in the directory textobjectfiles.

10. maketrees <terrain #><terrain name>

Maketrees reads in the treecorver and citycover files, compares them with xxx.ele to insure that the trees and cities are not on the roads. Then the program creates seven city and seven tree files to store the modified information.

11. genblockcov <terrain #><terrain name>

This program creates one kilometer by one kilometer grid square text files containing polygon descriptions. These files are stored in the directory textcoverfiles.

12. conv_blockcov2bin <terrain #><terrain name>

Conv_blockcov2bin converts the text files created by genblockcov and converts then to binary format. The new files are stored in the coverfiles directory.

13. genquadcov <terrain #><terrain name>

This program reads in the files created by conv_block2bin and places then into a quadtree structure. This files are then stored in the textquadfiles directory.

14. conv_quadcov2bin <terrain #><terrain name>

The textquadfiles are converted into binary format by this program and then are stored in the quadfiles directory located in the terrain specific directory.

15. genblockobj <terrain #><terrain name>

Genblockobj creates the tree and city canopies for the terrain. This files are in text form and placed in the textobject directory.

16. conv_block_obj_to_bin <terrain #><terrain name>

The textobjectfiles created by the program genblockobj are converted into binary format by this program and then are stored in the objectfiles directory located in the terrain specific directory.

How to Use the 3D Visualizer

Getting Started

Prior to running the program for the first time you need to change the file **units.dat** located in the **datafiles** directory. This file contains the names and telephone numbers of the units that can be called via the modem. There can be a maximum of nine units and phone numbers in the file. The names can be a maximum of six characters or letters on a line by itself. The telephone number associated with that unit should be on the following line. The telephone number can consist of a maximum of twenty numbers. e.g.:

```
199INF
17032212935
```

To execute the program, you need to be in the **visualizer** directory on a Silicon Graphics machine. At your unix prompt, type **janus3d NTC [enter].** (The terrain name can be substituted by any of the terrains you have in the terrain directory).The initial screen will be displayed with the credits. Note that at the lower center of the screen, information will be displayed as the different data files are read into the program. Once the program is finished loading, the working screen will appear. (See Figure A-2) With the main screen up, start Janus(A) 3.17 running. By starting the visualizer running first, when Janus initializes it's screens all the initial positions of the janus units will be transferred and displayed. Prior to beginning a scenario, remove all script files from the terrain directory. Otherwise, the new files will be appended to the old files.

There are three main sections to the display: the 3D view, 2D map and the vehicle information panel. The largest area is the 3D view. At initalization you are in the stealth mode. Through the use of the keyboard you can move freely throughout the battlefield. This area will display the world from your reference point. The other option is to be tethered to a vehicle. In this case, the 3D view will be from the vehicle's position.

The blue rectangle is the information panel. The panel contains the buttons to call other units, change the two dimensional map display, read scripted files, and stealth / janus vehicle information. This gives the user a numerical reference to where you are on or above the battlefield, the directions of travel and view, speed, and vehicle orientation, ID number and type. Direction is based on 0 degrees equates to North.

The lower right hand corner is the 2D map. The vehicles are iconized and color coded to make identification easy. The location of the icon is it's location on the battlefield. The line originating from each of the icons is the direction of travel with the length signifying the speed. (Longer lines indicate higher speeds). The yellow circle is your current location, while the green 'V' is the area of the map shown on the 3D display; the field of vision.

Moving in the Visualizer

There are two modes: tethered and stealth. While in the tethered mode, the user can change the viewing direction to the left or right and up or down. The direction and speed of the vehicle are determined by Janus(A) 3.17 running on the Hewlett Packard or a script of a previously run battle scenario. The stealth mode allows the user to move freely throughout the battlefield with all movements determined by the user through the keyboard.

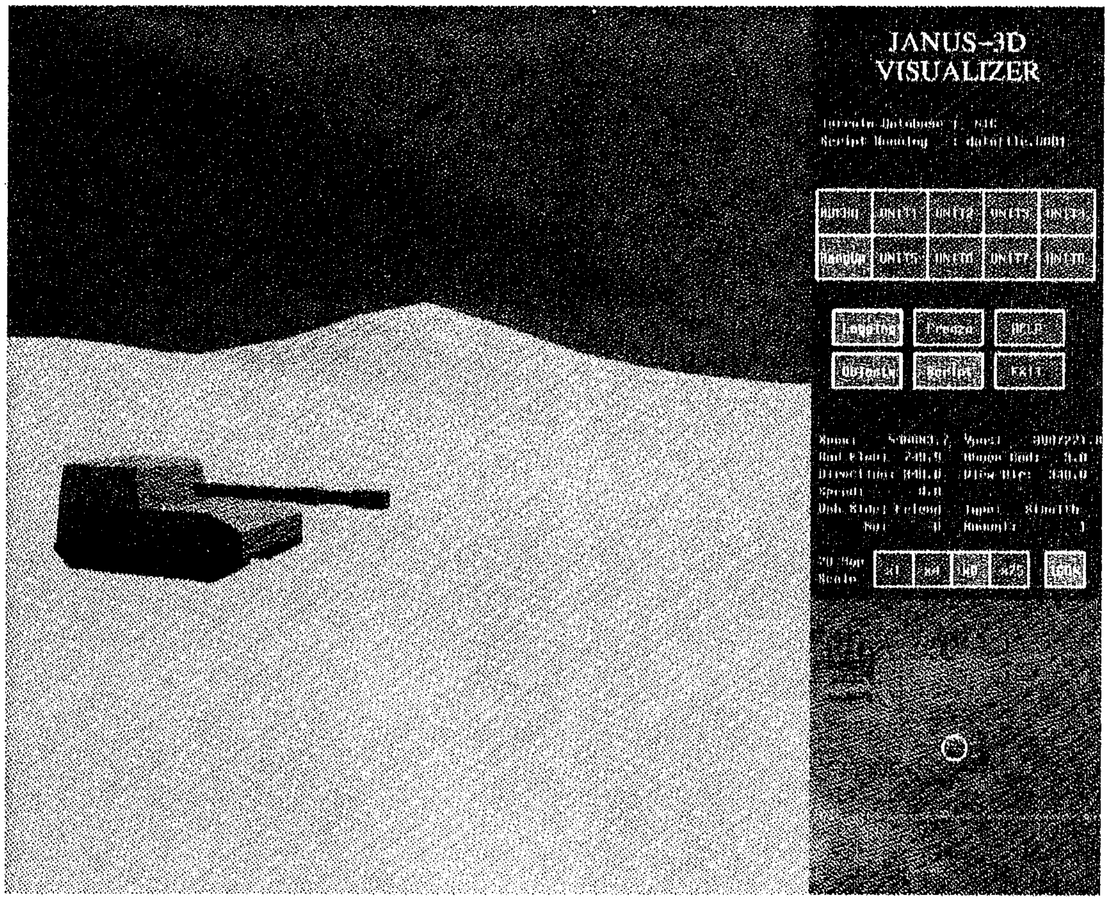


Figure A-2. Main Screen

Using the mouse

Left Mouse Button: Select a vehicle to tether on - Place the mouse cursor on a vehicle's icon or number in the 2D map area and click the left mouse button. The 3D screen will display what the selected vehicle can see in it's current direction of travel.

Middle Mouse Button: Untether- To untether from a vehicle, click the middle mouse button and the user will be in the stealth mode with the same view as from the vehicle that was deselected.

Right Mouse Button: Select Menu - Press the right mouse button while anywhere on the screen and the popup menu will appear. From this menu, you can take a picture of the screen.The image will be stored in the visualizer directory as snapshot#. The # will increase for each image stored during a session.

Using the keyboard

Tethered mode.

The **pad left arrow key** will move the field of view to the left.
The **pad right arrow key** will move the field of view to the right.
The **pad up arrow key** will allow the user to look up.
The **pad down arrow key** will allow the user to look down.
The **pad 5 key** will reset the view to the direction of travel of the tethered vehicle.

Stealth mode.

The **left arrow key** will change the direction of travel to the left.
The **right arrow key** will change the direction of travel to the right.
The **up arrow key** will increase the speed of the stealth vehicle.
The **down arrow key** will decrease the speed of the stealth vehicle.
The **end key** will stop the stealth vehicle.
The **pad left arrow key** will move the field of view to the left.
The **pad right arrow key** will move the field of view to the right.
The **pad up arrow key** will allow the user to look up.
The **pad down arrow key** will allow the user to look down.
The **pad 5 key** will reset the view to the direction of travel of the stealth vehicle.
The **page up key** will increase the elevation of the stealth vehicle.
The **page down key** will decrease the elevation of the stealth vehicle.

Information panel (Figure A-3)

The information panel contains the buttons to interact with the program and displays pertinent information about what is currently occurring in the program. The two lines under the title let the user know what terrain was loaded and, if they are running, a script and which script is running. The other non-interactive section of the panel is the vehicle information. The user is given the x and y grid coordinates of the vehicle, the ground elevation and elevation above ground of the stealth or tethered vehicle above the ground, the direction of travel, direction that the vehicle is looking and the speed of the vehicle are displayed. The last information displayed is the side (Friend or Hostile), the

Janus vehicle number, the Janus name from the master list located in datafiles/janusveh.dat (Type), and the number of systems that the icon and 3d model represent (Amount).

Calling another unit.

To call a unit place the mouse cursor in the box containing the name of the unit you wish to call, then press the left mouse button. The box will turn green and the program will try to establish a connection with that unit. Once a connection is established, current PDUs will immediately be transferred to the calling unit from the remote unit.To terminate the connection, place the mouse cursor in the Hang Up box, press the left mouse button. The Hung Up button will turn green and a box will appear asking if you really want to hang up. Select the OK button to terminate the connection or the No button to hide the box. If the OK button is selected, the box will automatically disappear once the connection has been terminated.

Logging

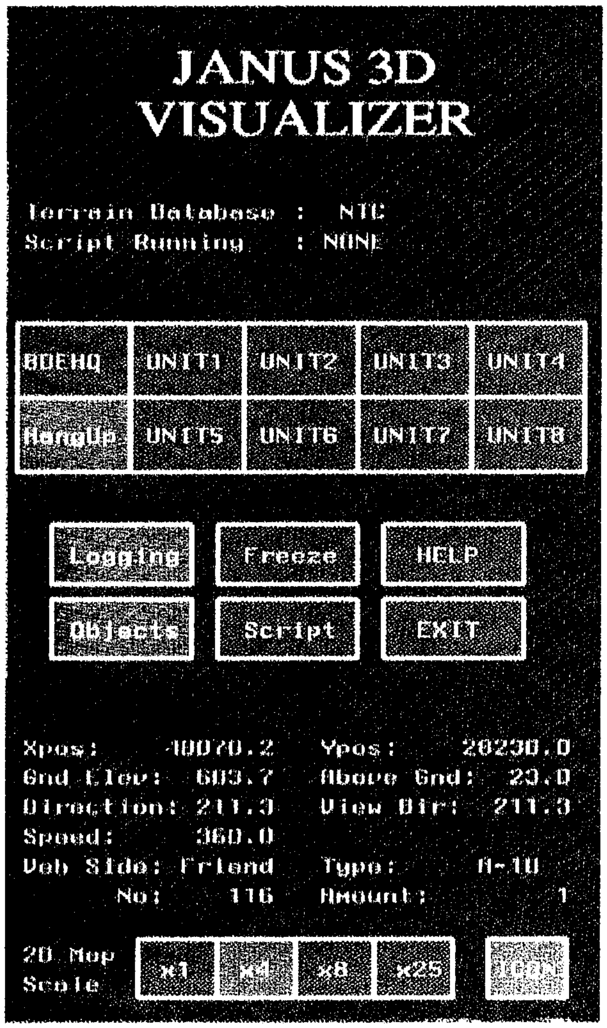
The logging button default is on (green). This will cause the program to create and store script files in the terrain directory. The program will not create a script file for the script files you are running or for the information displayed from another unit. If you do not want to create script files, move the cursor into the logging box and press the left mouse button. The box will turn blue to indicate that the script files are not being stored.

Freeze

The Freeze button only works when running a script file. This will stop the scripted vehicles from moving while allowing the stealth vehicle to travel around the frozen battlefield. To freeze the scripted file, move the mouse cursor into the Freeze box and press the left mouse button. The box will turn green indicating the script is frozen. To unfreeze, repeat the above procedure and the button will turn blue.

Help

When selected, the help button will display the keyboard inputs to move around the battlefield. To select the help menu, move the mouse cursor into the help box and press



```
Terrain Database  :  NTC
Script Running    : NONE

Xpos:        40070.2   Ypos:       20230.0
Gnd Elev:     603.7    Above Gnd:    23.0
Direction:   211.3     View Dir:    211.3
Speed:        360.0
Veh Side: Friend       Type:        A-10
      No:      116     Amount:          1
```

Figure A-3. Control Panel

the left mouse button. This will cause a large box containing the help information to appear in the information panel. When you are done looking at the help information, select the OK button and the box will disappear.

Objects

The Objects button default is on (green). With the objects selected, the trees and urban areas will be displayed. To remove the trees and urban areas move the mouse courser into the objects box and press the left mouse button. To redisplay the objects, repeat the above procedure and the box will turn green.

Script

To run a script, move the mouse cursor into the script box and press the left mouse button. A box containing a maximum of twenty script files will appear. Move the mouse cursor to the script file you want to run and press the left mouse button. The box will disappear and the name of the script file will appear at the top of the information panel. Each of the script files are twenty minutes long. Before each session, remove the old script file. This will keep the number of files to a minimum.

Exit

To exit the visualizer, move the mouse cursor into the Exit box and press the left mouse button. An exit box will appear and ask if you want to exit the program. Select the appropriate button to terminate the program.

Map Scale

The 2d map is initially set to x1. This displays the entire terrain file. x4,x8, x25 displays one fourth, one-eight, and one-twentyfifth of the map respectively (centered on your location). To change the map scale move the mouse courser into the box containing the desired scale and press the left mouse button.

Icon/Num

This changes the 2d icon display. To change from the default icon setting to the Janus number move the mouse cursor into the icon/numeric box and press the left mouse button. By repeating this process you can toggle between number and icon in the 2dmap.

2D Display

The map gives the user a gray scale elevation representation of the terrain. Black is the low ground and white is the high ground. The grid squares on the map represent one kilometer grid squares, Figure A-4.

The vehicles can be depicted as numerics or as icons. The numbers range from one to six hundred for both forces. The icons / numerics for the friendly forces are blue, the enemy forces are red, and the dead vehicles are green. The user's location is indicated by a yellow circle. The green triangle extending from the circle is the field of view that is displayed in the 3D window.

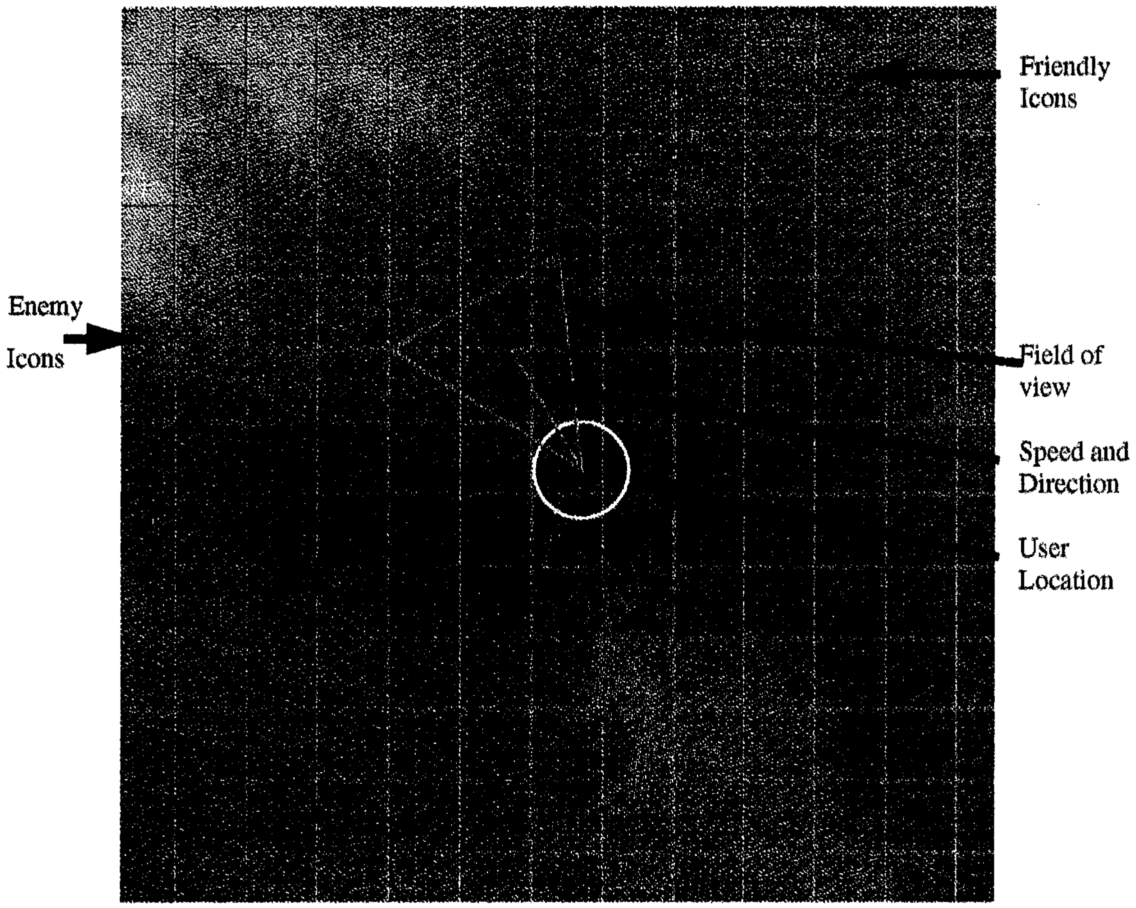


Figure A-4. 2D Map Display

LIST OF REFERENCES

[FUNK94] Funk, Steven, *Information Paper ARPA/ARNG Advanced Technology Demonstration #2 Project SIMITAR*, Ft. Leavenworth, KS.

[INST91] Institute for Simulation and Training, *Military Standard: Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation*, Institute for Simulation and Training, Orlando, FL.

[JANU93] Department of Army, *The Janus 3.X/UNIX Model Software Design Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JANU93b] Department of Army, *The Janus 3.X/UNIX Model System Design Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[JANU93c] Department of Army, *The Janus 3.X/UNIX Model User's Manual*, Headquarters TRADOC Analysis Center, ATRC-ZD, Ft. Leavenworth, KS, May 1993.

[MORN91] Morningstar, Chip, and Farmer, F. Randall, *The Lessons of Lucasfilm's Habitat* in Cyberspace First Steps, ed. Benedikt, Michael, The MIT Press, Cambridge, 1991.

[NAKA94] Nakamura, Nobutatsu, Nemoto, Keiji, and Shinohar, Katsuya, *Distributed Virtual Environment System for Cooperative Work*, 1st International Workshop on Networked Reality, '94 Proceedings, May 1994.

[PRAT93] Pratt David R., *A Software Architecture for the Construction and Management of Real-Time Virtual Worlds*, Dissertation, Naval Postgraduate School, Monterey, CA, June 1993.

[PRAT94] Pratt David R., and Locke, John, *A Virtual Reality Interface for Real World Weapons Testing*, Naval Postgraduate School, Monterey, CA, July 1994.

[USRO92] U.S. Robotics, Inc., *USRobotics Sportster 9600 and Sportster 9600 Fax User's Guide*, U.S. Robotics, Inc., Skokie, IL, 1992.

[VAGL94] Vaglia, James A., *Creating a Real-Time Three Dimensional Display for the Janus Combat Modeler*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1994.

[WALT92] Walter, Jon C., and Warren, Patrick T., *NPSNET: Master's Thesis in Computer Science, JANUS-3D Providing Three-Dimensional Displays for a Traditional Combat Model*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1992.

[ZESW93] Zeswitz, Steven R., NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Interchange, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1993.

[ZYDA92] Zyda, Michael J., Pratt, David R., Monahan, Gregory, and Wilson, Kalin P., *NPSNET: Constructing a 3D Virtual World*, Symposium on 3D Graphics, '92 Proceedings, April 1992, pp 147-156.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 052 Naval Postgraduate School Monterey, CA 93943	2
3.	Chairman Ted Lewis, Code CS/Lt Computer Science Department Naval Postgraduate School Monterey, CA 93943	2
4.	Professor D. R. Pratt, Code CS/Pr Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	2
5.	Professor G.M. Lundy, Code CS/Ln Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	2
6.	MAJ Tom Allen ARPA-SIMITAR Fort Leavenworth, KS 66027	1
7.	Mr. Don Bennett Cubic Applic Inc. P. O. Box 13548 Fort Carson, CO 80913	1
8.	Mrs. Meg Champion LTSI Box 1825 Richmond Hill, GA 31324	1

9.	Mr. Jeffrey K. Skilling BDM Federal Inc. P. O. Box 908 Fort Knox, KY 40121	1
10.	CPT Christopher S. Upson RD#3, Box 297 Frankfort, NY 13340	3